НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.89

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти»

Виконав:
студент ІІ курсу, групи КП-81мп
Редін Кирило Артурович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА, к.ф.-м.н., доцент,
Дідковська Марина Віталіївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітня програма) – 121 «Інженерія програмного забезпечення» ("Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем")

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студентці

Рединy Кирилу Артуровичу

1. Тема дисертації «Комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «13» листопада 2019 р. № 3895-С.

2. Термін подання студентом дисертації «16» грудня 2019 р.

3. Об'єкт дослідження: фреймворки для створення крос-платформних користувацьких додатків.

4. Предмет дослідження: способи інтеграції з "рідними" API мобільних платформ та архітектура.

5. Перелік завдань, які потрібно розробити:
- оцінити сучасний стан проблеми, обґрунтувати актуальність напрямку досліджень, сформулювати мету та задачі дослідження;
- описати сферу розповсюдження повідомлень електронної пошти;
- сформувати вимоги до методу класифікації текстового вмісту повідомлень електронної пошти під час аналізу існуючих рішень для класифікації текстових даних;
- описати принципи, що використовуються у комбінованому методі автоматизованої класифікації: описати їх особливості та детально описати розділення етапів фільтрації та класифікації, а також модифікацію метода Баєса;
- описати архітектуру системи в якій реалізовано комбінований метод класифікації, структуру клієнтської та серверної частини;
- виконати тестування розробленого програмного забезпечення комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти та проаналізувати отримані результати;
- описати бізнес-модель, що дозволить представити на ринку повноцінний програмний продукт із використанням представлених у роботі напрацювань.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - загальна архітектура підходу;
   - схема організації крос-платформного тексту програми у розроблюваній системі;
   - діаграма класів каналу комунікації;
   - результати аналізу розробленого програмного забезпечення;
   - дерево проблем та рішень.

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Комбінований метод автоматизованої класифікації повідомлень електронної пошти”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		

9. Дата видачі завдання «25» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	16.11.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	07.12.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	13.02.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	08.04.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	14.05.2019	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	19.06.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	09.11.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	07.12.2019	

Студент К.А. Редін

Науковий керівник дисертації Т.М. Заболотня

РЕФЕРАТ

Актуальність теми. З появою мережі Інтернет великої популярності здобули електронні поштові системи. Великі обсяги природномовних текстових даних електронної пошти, а також висока швидкість їх надходження потребують застосування спеціальних методів автоматизованої класифікації. З кожним роком зростає кількість користувачів електронних поштових систем. Разом з цим збільшується середня кількість повідомлень, яку користувач отримує щодня, а також відсоток спам-повідомлень. Це підвищує інформаційну “забрудненість” електронної поштової скриньки, що негативно впливає на досвід її використання кінцевим користувачем. Усі ці факторі роблять дуже актуальною задачу розробки нових та вдосконалення існуючих методів автоматизованої класифікації текстового вмісту електронних повідомлень.

Об’єктом дослідження є процес класифікації природномовних текстових даних електронної пошти.

Предметом дослідження є методи автоматизованої класифікації текстових даних.

Мета роботи: підвищення точності автоматизованої класифікації текстового вмісту повідомлень електронної пошти за рахунок розробки та реалізації нового методу класифікації.

Методи дослідження. В роботі використовуються методи представлення текстових даних та методи машинного навчання.

Наукова новизна роботи полягає в наступному:

Запропоновано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти, що відрізняється від існуючих розділенням етапів фільтрації та класифікації за категоріями та дозволяє досягти підвищення точності класифікації.

Запропоновано модифікацію метода Баєса, що відрізняється від існуючого методу введеним граничним значенням ймовірності, а також логарифмічної оцінки ймовірностей та дозволяє досягти підвищення

точності класифікації та можливості класифікації текстового вмісту повідомлень за декількома категоріями.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований комбінований метод класифікації текстового вмісту повідомлень електронної пошти дозволив збільшити точність класифікації повідомлень завдяки розділенню етапів фільтрації та класифікації за категоріями, а також модифікації метода Баєса.

Розроблено веб-додаток для обміну текстовими повідомленнями електронної пошти, який реалізує запропонований метод класифікації, в якому враховано сучасні потреби до швидкості надходження електронних повідомлень та безпеки персональних даних користувача.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019 (Київ, 13–15 листопада 2019 р.) та опубліковані у збірнику тез за результатами конференції.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень.

У першому розділі проаналізовано існуючі рішення для класифікації текстового вмісту повідомлень електронної пошти. Виявлено основні переваги та недоліки існуючих рішень. Визначено найбільш ефективні методи для фільтрації повідомлень від спаму та їх класифікації за категоріями.

У другому розділі описано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти. Запропонований метод полягає в розділенні етапів фільтрації та класифікації. На етапі фільтрації використовується метод опорних векторів,

який розділяє повідомлення на два класи. На етапі класифікації використовується модифікований метод Баєса, який визначає ймовірність віднесення повідомлення до відповідних категорій. Модифікація методу Баєса полягає у використанні логарифмічної оцінки ймовірності для підвищення точності, а також у введенні граничного значення ймовірності для можливості класифікації повідомлення за декількома категоріями.

У третьому розділі обгрунтовано вибір технологій для розробки веб-додатку, який реалізовує запропонований метод. Описана загальна архітектура системи. Розглянуто структуру клієнтської та серверної частини. Визначено функціональні можливості розробленого веб-додатку.

У четвертому розділі було проведене тестування реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти, а також порівняно їх з результатами інших методів класифікації. За результатами тестування виявлено, що реалізований метод має більшу точність класифікації, в порівнянні з існуючими рішеннями, а також має досить високу для класифікації в реальному часі швидкість.

У п’ятому розділі проаналізовано сферу електронних поштових систем. За виконаним аналізом побудовано бізнес-модель для представлення структурних, операційних та фінансових механізмів роботи для створення веб-додатку, що реалізує запропонований метод.

У висновках проаналізовано отримані результати роботи.

У додатках наведено фрагменти програмної реалізації запропонованого методу та копії графічних матеріалів

Робота виконана на 90 аркушах, містить 2 додатки та посилання на список використаних літературних джерел з 16 найменувань. У роботі наведено 35 рисунків та 10 таблиць.

Ключові слова: класифікація тексту, машинне навчання, електронна пошта, фільтрація спаму, класифікація за категоріями, метод Баєса, метод опорних векторів.

ABSTRACT

Actuality. With the advent of the Internet, email systems have become very popular. Large volumes of natural-language text-based email data, as well as high speed of their receipt, require the use of special methods of automated classification. The number of users of email systems is increasing every year. At the same time, increases the average number of messages that user receives each day and the percentage of spam messages. It causes large amount of unnecessary information in email box, which affects the users experience. All these factors make it very important to develop new and improve existing methods for automated email text content classification.

Object of research is the process of automated classification of email text data.

Subjects of research are methods of automated classification of text data.

Goal of the work is to improve accuracy of the automated email text data classification by developing and implementing a new classification method.

Methods of research. Textual data representation and machine learning methods are used in the work.

Scientific novelty is as follows:

A combined method for automated email text data classification is proposed, which differs from the existing methods by separation of filtering and classification stages. It allows to increase the classification accuracy.

A modification of the naive Bayes method is proposed, which differs from the existing method by the introduced probability threshold, as well as the logarithmic estimation of probabilities. It improves the accuracy of classification and the allows to classify email text data by several categories.

Practical value of the obtained results is that proposed combined method for automated email text data classification increases the accuracy of classification of messages by separating filtering and classification by categories stages and naive Bayes method modification.

A web-based email messaging application which implements the proposed classification method has been developed. All current needs for an email sending speed and the security of user personal data were taken into consideration.

Approbation. The main provisions and results of the work were presented and discussed at the XII scientific conference of masters and postgraduates "Applied Mathematics and Computer" PMK-2019 and published in the proceedings.

Structure and content of the thesis. Master's thesis consists of an introduction, five chapters, conclusions and appendices.

The introduction provides a general description of the work, evaluated the current state of the problem, substantiated the relevance of the research direction, formulated the purpose and objectives of the study.

In the first chapter existing solutions for automated email text data classification. The main advantages and disadvantages of existing solutions have been identified. The most effective methods for messages filtering and classification were identified.

In the second chapter combined method for automated email text data classification was purposed. In this method stages of email filtering and classification were separated. The filtering method uses the support vector machine method, which classifies message by two categories. On the classification stage, a modified naive Bayes method is used. It which determines the probability of message being assigned to the relevant category. In modified Bayes method logarithmic estimate of probability was used to improve accuracy and probability threshold was introduced to classify message by multiple categories.

In the third section, the main technologies for developing a web application that implements the proposed method were justified. The general architecture of the system was described. The structure of client and server part was considered. The functionalities of the developed web application have been determined.

In the fourth chapter implementation of a combined method for automated email text data classification was tested. Results of the tests were compared with results of previous methods. Was proved that proposed method is more accurate and has same classification speed as others method.

In the fifth chapter the scope of email systems has been analyzed. A business model was created for the representation of structural, operational and financial mechanisms for the development of web application that implements proposed method.

The conclusion contains brief overview of the results obtained in the work.

The work is done on 90 pages, contains 2 appendices and reference list of 16 titles. The work contains 35 pictures and 10 tables.

Keywords: text classification, machine learning, email, spam filtering, categorization, naive Bayes, support vector machine.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 4
ВСТУП 5
1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ 6
1.1. Класифікація на основі правил 6
1.2. Класифікація на основі ваг 7
1.3. Методи машинного навчання 8
1.4. Нейронні мережі 15
1.5. Порівняння існуючих рішень 22
1.6. Висновки розділу 1 25
2. КОМБІНОВАНИЙ МЕТОД АВТОМАТИЗОВАНОЇ КЛАСИФІКАЦІЇ ТЕКСТОВОГО ВМІСТУ ЕЛЕКТРОННОЇ ПОШТИ 26
2.1. Етап тренування 27
2.2. Попередня обробка даних 31
2.3. Етап класифікації 34
2.4. Переваги запропонованого методу 39
2.5. Висновки до розділу 2 39
3. ПРОГРАМНА РЕАЛІЗАЦІЯ 41
3.1. Вибір технологій для вирішення поставленої задачі 41
3.2. Архітектура розробленої системи 47
3.3. Опис функціональних можливостей системи 62
3.4. Висновки до розділу 3 64
4. ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ 66
4.1. Тестування спам класифікатора 67
4.2. Тестування класифікатора за категоріями 70
4.3. Тестування комбінованої класифікації 71
4.4. Висновки до розділу 4 73
5. ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ 74
5.1. Опис проблеми 74
5.2. Зацікавлені сторони 77

5.3. Комерційне рішення. Основні характеристики 79
5.4. Конкурентні переваги рішення 80
5.5. Клієнти. Сегменти ринку споживання 82
5.6. Унікальна ціннісна пропозиція 83
5.7. Доходи та витрати 84
5.8. Бізнес модель 88
5.9. Висновки до розділу 5 90
ВИСНОВКИ 91
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ 92
ДОДАТКИ 94

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Електронна пошта – спосіб обміну електронними повідомленнями між користувачами мережі Інтернет за допомогою використання електронних пристроїв.

Спам – масова розсилка повідомлень електронної пошти рекламного характеру особам, які не мають бажання її отримувати.

Задача класифікації – задача, віднесення об’єкта до одного або декількох класів із заздалегідь визначеної множини.

Задача кластеризації – задача розбиття множини об’єктів на групи, які називаються кластерами.

Задача відновлення регресії – задача відновлення функції умовного математичного очікування.

Машинне навчання – підрозділ методів штучного інтелекту, особливістю яких є навчання на множині рішень аналогічних задач

Штучна нейронна мережа – математична модель, яка являє собою систему поєднаних штучних нейронів, які взаємодіють між собою.

Штучний нейрон – спрощена модель нейрона, яка являє собою нелінійну функцію.

Мішок слів (bag-of-words) – спрощена модель представлення текстових даних, в якій текст конвертується в словник, де кожне слово позначається відповідним індексом.

Лематизація – процес приведення слова до його нормальної форми.

Стемінг – процес скорочення слова до основи шляхом відкидання допоміжних частин.

Аутентифікація – процес перевірки ідентифікатора пред’явленого користувачем для встановлення належної йому інформації.

Об’єктно-реляційне представлення – технологія програмування, яка пов’язує бази даних з концепціями об’єктно-орієнтованих мов програмування

ВСТУП

З появою та розвитком мережі Інтернет великої популярності набула електронна пошта. Швидкість передачі електронних повідомлень зробили її незамінною спочатку у бізнес-сфері, а згодом і серед звичайних користувачів. Щороку кількість зареєстрованих електронних скриньок зростає, але із зростанням популярності електронної пошти збільшується і масштаб проблем, що з'явилися з її появою.

Однією з основних проблем електронної пошти є спам-повідомлення.

За статистикою у 2019 році бізнес-користувач електронної пошти отримує в середньому 96 електронних повідомлень на день, близько 20% з яких є рекламою або спамом [1]. За останні 4 роки кількість спаму виросла майже на 10%. Очевидно, що в майбутньому кількість повідомлень буде лише зростати, але, разом з тим, вже зараз існує потреба у їх класифікації для фільтрування від спаму.

Всі повідомлення, що минують спам-фільтр, потрапляють до однієї купи. Користувачу електронної пошти важко знайти необхідне електронне повідомлення серед десятків інших, оскільки відсутня класифікація їх за категоріями.

На сьогоднішній день існують методи для фільтрації та класифікації повідомлень електронної пошти за категоріями. Усі вони є недостатньо ефективними, оскільки мають досить низьку точність через використання єдиного класифікатора для фільтрації та класифікації за категоріями.

Отже, підвищення точності автоматизованої класифікації повідомлень електронної пошти є актуальною задачею.

Дана магістерська дисертація присвячена створенню комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти, а також розробці веб-додатку, який реалізує цей метод.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1. Класифікація на основі правил

До класифікації текстових даних на основі правил можна віднести усі схеми, які використовують правила if-then-else для віднесення документу до певної рубрики [2].

Зазвичай схеми класифікації на основі правил складаються з таких компонентів:

1. Алгоритм індукції правила. Процес формування правил з даних за допомогою алгоритму послідовного покриття або інших способів.
2. Ранжування правил. Процес виміру корисності правила для видалення необов’язкових правил та підвищення ефективності.

Метод класифікації на основі правил є дуже ефективним, якщо необхідно виконати класифікацію невеликого набору документів, який можна ретельно проаналізувати.

Основним недоліком методу є те, що для створення набору правил необхідно бути експертом в заданій предметній сфері. Крім того, якщо правила базуються на присутності або відсутності слова в документі, класифікація буде мати достатньо низьку точність, оскільки окремі слова рідко є ключовими факторами при класифікації. Будь-які спам-слова можуть зустрічатися і в звичайних повідомленнях, навіть якщо ймовірність цього дуже мала.

Крім того, чим більше різних категорій мають бути охоплені, тим складніші правила необхідно формувати. Оскільки такі системи потребують постійної підтримки, правила постійно редагуються та додаються. Це призводить до того, що з часом набори правил стають занадто складними для розуміння, редагування або додавання нових. При формуванні вкладених правил їх складність зростає експоненційно з кількістю слів, які обрані для класифікації.

Іншою проблемою є те, що такий метод є неповністю автоматизованим. Підтримка правил потребує постійної присутності одного

або декількох спеціалістів в предметній сфері. При необхідності додавання нових правил потрібен час на їх створення та оновлення, що є критичним в деяких системах.

Використання методу класифікації на основі правил є недоцільним для автоматизованої класифікації повідомлень електронної пошти. Існуючі недоліки роблять неможливим проведення автоматизованої класифікації в реальному часі без постійної участі спеціаліста. Крім того, використання даного методу не дозволяє досягти бажаної точності класифікації, яка становить не менше 80%.

1.2. **Класифікація на основі ваг**

Метод класифікації на основі ваг полягає в присвоєнні ваги усім властивостям об’єкту, який необхідно класифікувати [2]. У випадку повідомлень електронної пошти властивостями, які потрібно брати до уваги, є слова в повідомленні, а ваги задаються вручну спеціалістом в предметній галузі. Якщо спосіб визначення ваги автоматизований, цей метод буде відноситися до категорії методів машинного навчання.

На відміну від методу класифікації на основі правил, цей метод є більш гнучким, оскільки не однозначно вирішує, чи відноситься повідомлення до спаму за присутніми у ньому словами, а дозволяє визначити вагу повідомлення для кожного класу як добуток ваг слів цього повідомлення. У випадку, якщо слово не має своєї ваги, воно буде проігноровано.

Даний метод також надає можливість додавати нові ваги та модифікувати існуючі без впливу на складність класифікації.

Хоча метод класифікації на основі ваг позбавлений головних недоліків методу класифікації на основі правил, він все ще потребує постійної участі спеціаліста для підтримки ваг для існуючих класів та створення нових. В контексті повідомлень електронної пошти постійна модифікація

класифікатора вручну є неможливою, оскільки нові класи з’являються дуже часто, а кількість слів занадто велика для обробки вручну.

1.3. **Методи машинного навчання**

Наразі найпопулярнішим вирішенням задачі класифікації текстових даних є використання методів машинного навчання [7]. На рис. 1.1 зображені різновиди машинного навчання у вигляді схеми.

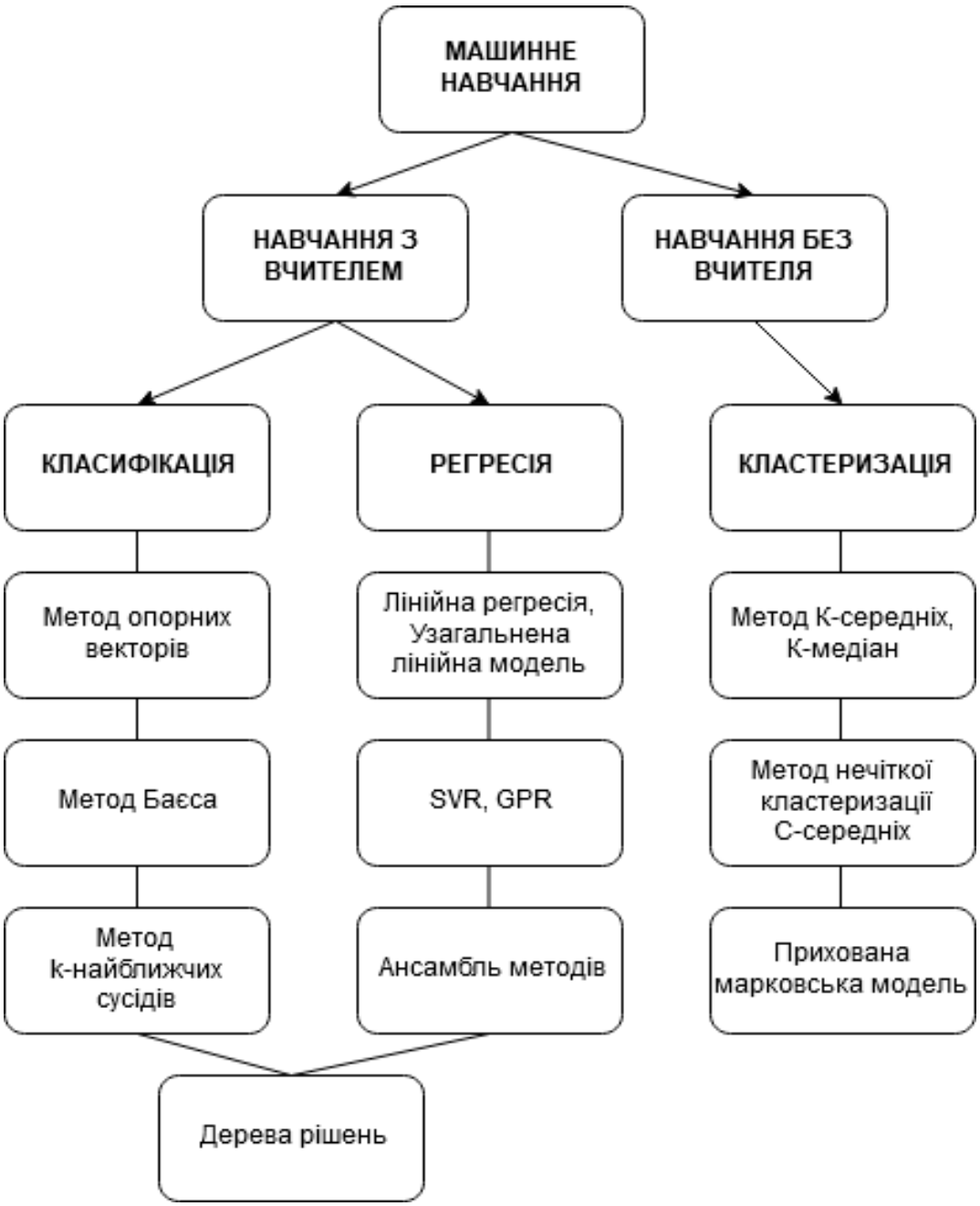


Рис. 1.1. Схема класифікації методів машинного навчання

Машинне навчання можна поділити на контрольоване та неконтрольоване [3]. До контрольованого, тобто навчання з вчителем, відносяться методи, що вирішують задачі класифікації та регресії, а до неконтрольованого – задачі кластеризації.

Методи, які вирішують задачі відновлення регресії, використовуються для передбачення числових значень, таких як ціна на квартиру, очікуваний прибуток, рейтинг енерговитратності, тому вони не можуть бути застосовані для класифікації повідомлень електронної пошти.

Хоча кластеризація також може бути застосована для аналізу електронної пошти, найбільш ефективними є методи, що вирішують задачу класифікації, оскільки в цьому випадку заздалегідь відомі класи, до яких необхідно віднести повідомлення [6].

*1.3.1 **Метод Баєса***

Одним з найбільш розповсюджених методів класифікації є метод Баєса. Цей метод базується на теоремі Баєса, яка дозволяє визначити ймовірність певної події [4].

При аналізі вмісту електронної пошти в якості документу виступає електронне повідомлення *d*, до якого відноситься вектор ознак $f = (f_1, \ldots f_n)$. Кожному документу ставиться у відповідність найбільш ймовірна категорія *c* з вектором ознак $c = (g_1, \ldots, g_n)$. Категорія визначається за формулою (1.1):

$$c^* = \text{argmax}_{c \in C} P(g_1 = f_1, \ldots, g_n = f_n) \quad (1.1)$$

Для визначення ймовірності віднесення документу до категорії необхідно включити в навчальну вибірку всі можливі комбінації документів та класів, що є неможливим. Через це необхідно переписати формулу наступним чином:

$$c^* = \text{argmax}_{c \in C} \frac{P(g_1 = f_1, \ldots, g_n = f_n | c) P(c)}{P(g_1 = f_1, \ldots, g_n = f_n)} \quad (1.2)$$

У формулі (1.2) можна не враховувати знаменник, оскільки він не залежить від *c* та не впливає на результат.

$$c^* = \text{argmax}_{c \in C} P(g_1 = f_1, \dots, g_n = f_n | c) P(c) \quad (1.3)$$

В результаті отримано формулу (1.3) для визначення категорії документу, де $P(c)$ – ймовірність того, що зустрінеться клас *c*, незалежно від документу, що розглядається, а $P(g_1 = f_1, \dots, g_n = f_n | c)$ – ймовірність зустріти документ серед документів класу *c*.

Метод Баєса застосовується окремо до кожної категорії, визначаючи, чи відноситься до неї документ, що розглядається. Це є перевагою для вирішення задачі класифікації електронної пошти, оскільки надає можливість відносити одне повідомлення до декількох класів одразу.

Недоліком цього методу є припущення про незалежність ознак *g1, . . . , gn*, яке не виконується в більшості випадків. Але, не зважаючи на це, при класифікації тексту метод демонструє достатньо високі результати.

Необхідно наголосити на тому, що метод Баєса, завдяки високій швидкості навчання, дозволяє проводити класифікацію в режимі реального часу, що є дуже важливим фактором при класифікації електронної пошти, яка має проходити досить швидко, щоб бути непомітною для користувача.

Для більш точної класифікації електронної пошти цей метод доцільно застосовувати в комбінації з іншим методом, який буде проводити попередню фільтрацію від спаму. Це не вплине на швидкодію клієнтської частини, оскільки класифікація документу відбувається досить швидко, а найбільш витратним за часом є саме тренування класифікаторів, яке відбувається один раз на сервері, незалежно від запитів користувача.

*1.3.2 **Метод опорних векторів***

Інший популярний метод класифікації даних, метод опорних векторів, базується на побудові гіперплощини, яка розділяє простір з документами на дві частини [5].

Основною задачею методу є визначення гіперплощини, для якої відстань до найближчих прикладів є найбільшою.

У випадку задачі класифікації електронної пошти X буде простором повідомлень, а C – класами $\{-1, 1\}$. Розділяюча гіперплощина буде мати вигляд $w*x+b=0$, де w – перпендикуляр до гіперплощини, а b – відстань по модулю від гіперплощини до початку координат. Найближчі до гіперплощини повідомлення і будуть опорними векторами.

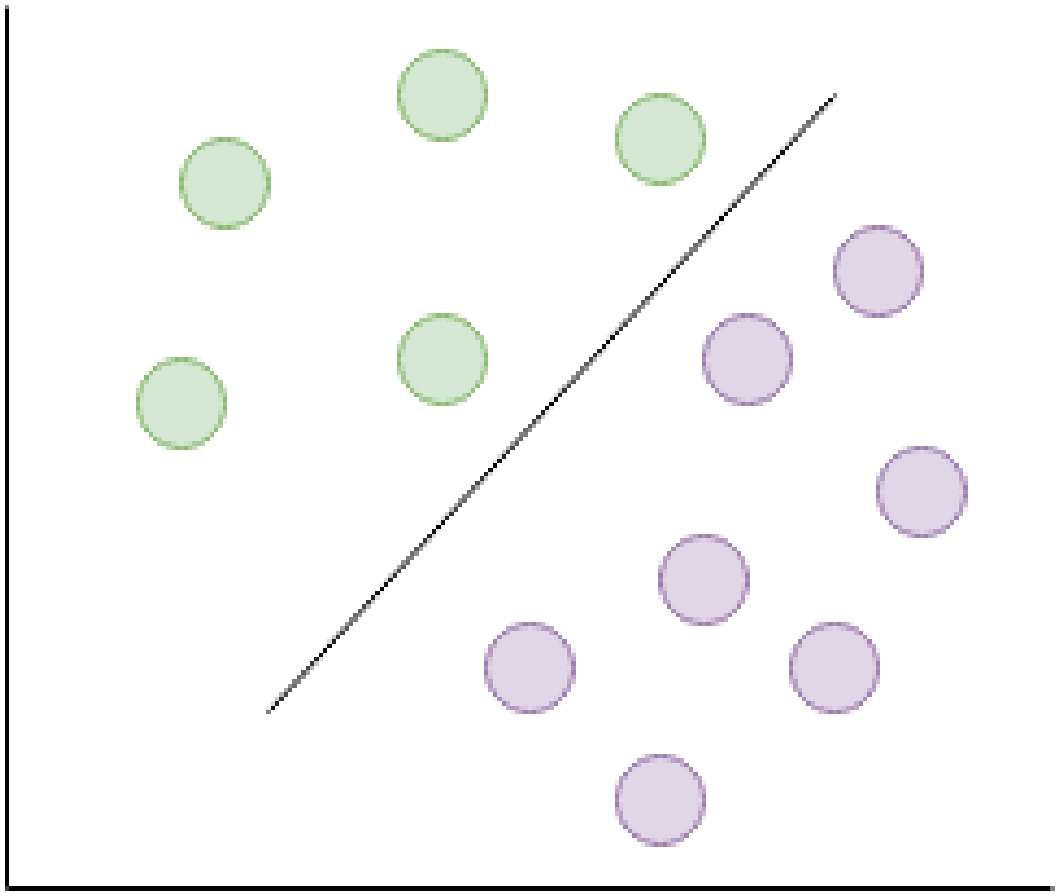


Рис. 1.2. Схема розділення об’єктів на класи

Цей метод розділяє документи гіперплощиною на два класи (рис. 1.2), тому його доцільно застосовувати для фільтрації спам-повідомлень.

Через те що при фільтрації електронної пошти використовується лише два класи, достатньо використовувати лінійне ядро. В цьому випадку класифікація досить швидка, завдяки чому метод є ефективним для використання в реальному часі.

Інші типи ядер є надлишковими для розділення документів на два класи та потребують більше обчислювальних ресурсів, тому їх використання в методі опорних векторів для фільтрації електронної пошти від спаму є недоцільним.

1.3.3 Метод k-найближчих сусідів

Наступним методом класифікації є метод k-найближчих сусідів. Ідея цього методу полягає в пошуку найбільш релевантних рубрик для нового документу [3].

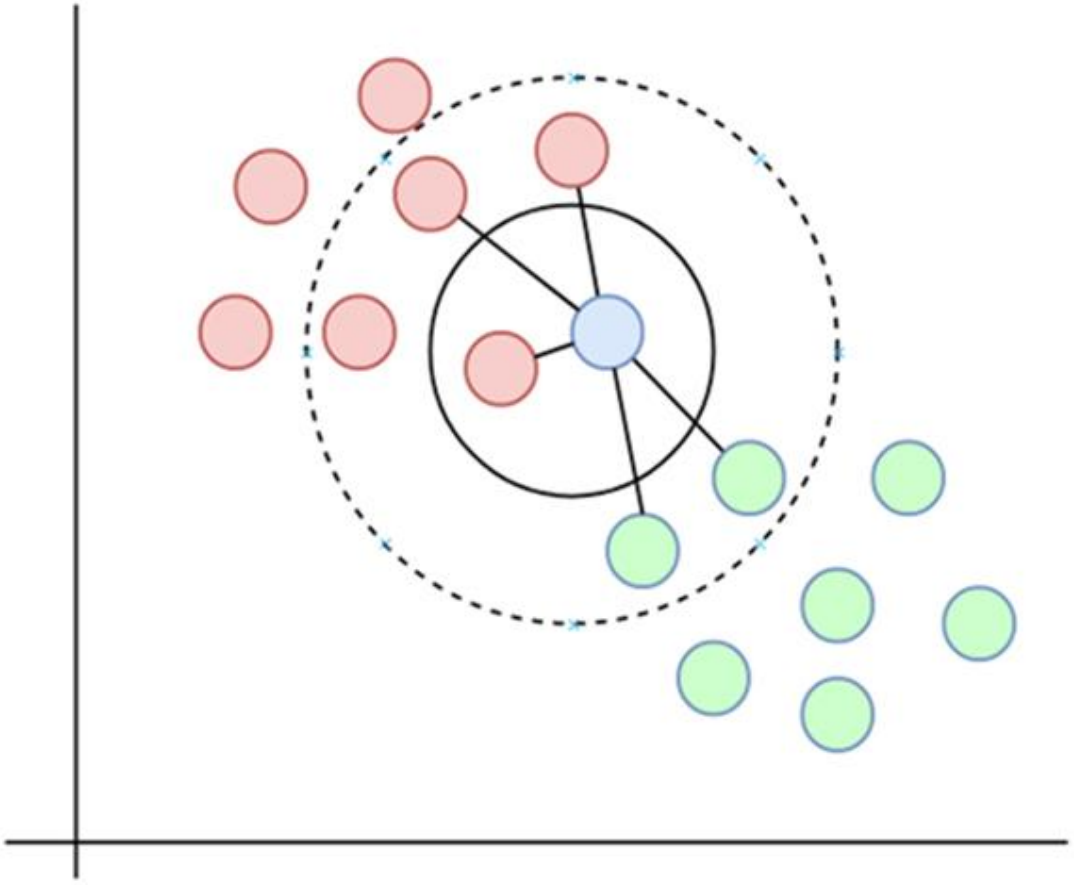


Рис. 1.3. Схема k-найближчих до об’єкту сусідів

На першому етапі повідомлення d порівнюється з усіма повідомленнями d_e, які належать навчальній виборці. Потім знаходиться відстань до усіх повідомлень за формулою $p(d, d_e) = \cos(d, d_e)$.

Після цього обирається k найближчих повідомлень та обчислюється релевантність для кожної рубрики за наступною формулою:

$$s(c_j, d) = \sum_{d_e \in Ex_k(d)} \cos(d, d_e) \quad (1.4)$$

В формулі (1.4) $Ex_k(d)$ — k найближчих до d повідомлень з навчальної вибірки. Повідомлення в даному випадку відноситься до найбільш релевантних рубрик.

Недоліком є те, що метод k-найближчих сусідів ресурсо-витратний, оскільки кожного разу при додаванні нового повідомлення необхідно перераховувати релевантність усіх повідомлень, що робить неможливим його застосування у випадку швидкого надходження нових повідомлень.

*1.3.4 **Дерева рішень***

Одним з найпопулярніших методів, який використовується для вирішення задач класифікації та регресії, є дерево рішень.

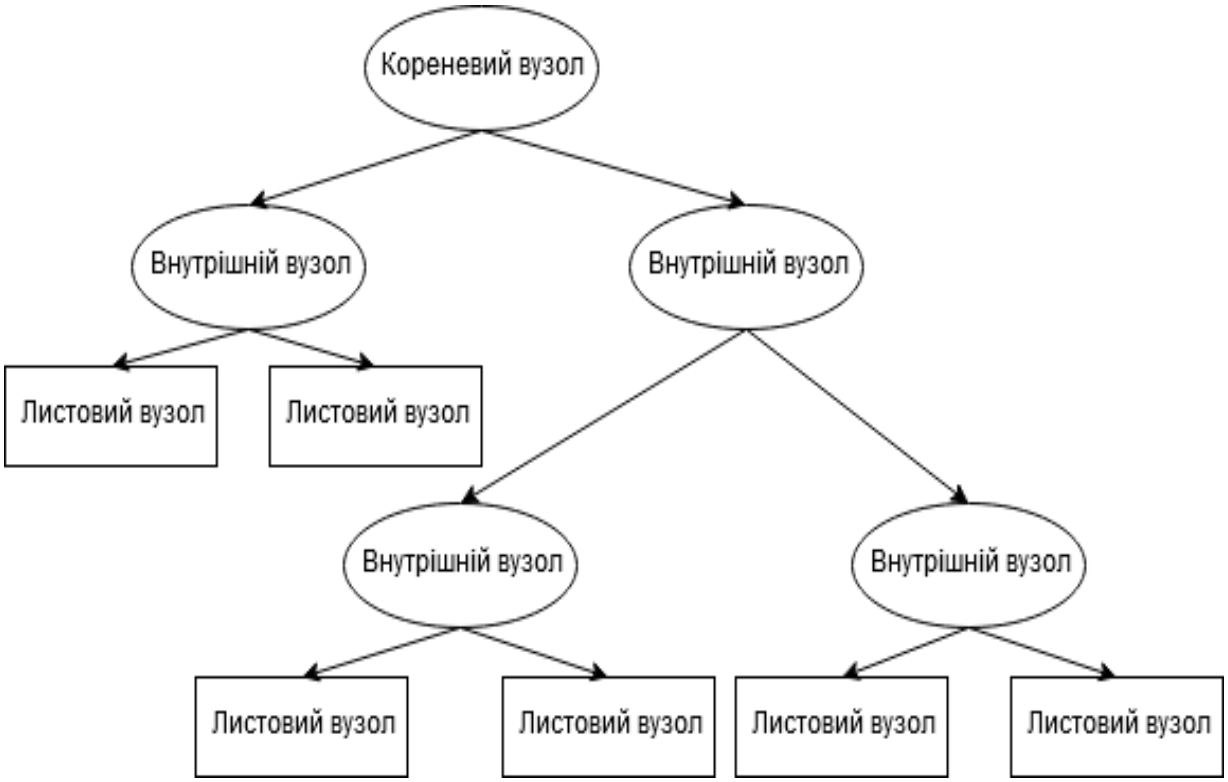


Рис. 1.4. Структура дерева рішень

Цей метод полягає в побудові дерева, кожним вузлом якого є предикат, що визначає, по якій саме гілці “прослідує” повідомлення під час класифікації [7].

Спускаючись деревом, повідомлення потрапляє у листовий вузол, який є кінцевим класом повідомлення. На рис. 1.4 зображена структура дерева рішень.

Хоча за допомогою дерева рішень кожне повідомлення попадає лише до одного класу, можна знайти ймовірність потрапляння повідомлення до інших гілок.

На жаль, через структуру дерева рішень повідомлення в будь-якому випадку буде віднесено до певного класу. Це робить їх не найкращим рішенням для класифікації електронної пошти, оскільки повідомлення можуть не відноситись до жодного класу.

1.3.5 ***Випадковий ліс***

Випадковий ліс є одним з класичних методів машинного навчання. Цей метод використовують для вирішення задач відновлення регресії, кластеризації, класифікації, тощо [7].

Випадковий ліс складається з множини дерев рішень. На кожне з дерев подаються дані для класифікації, у випадку повідомлень електронної пошти це числові вектори ознак текстового вмісту повідомлення. Ознаками повідомлення є кількість зустрічей кожного слова в конкретному повідомленні.

Кожне з дерев відносить повідомлення до одного з класів, після чого проводиться “голосування”, в ході якого повідомлення відноситься до класу, за який “проголосувало” найбільше дерев. На рис. 1.5 зображена структура випадкового лісу.

Хоча випадковий ліс надає більшу точність класифікації, як і дерева рішень, він позбавляє можливості не класифікувати повідомлення зовсім, що робить його неефективним для класифікації за категоріями.

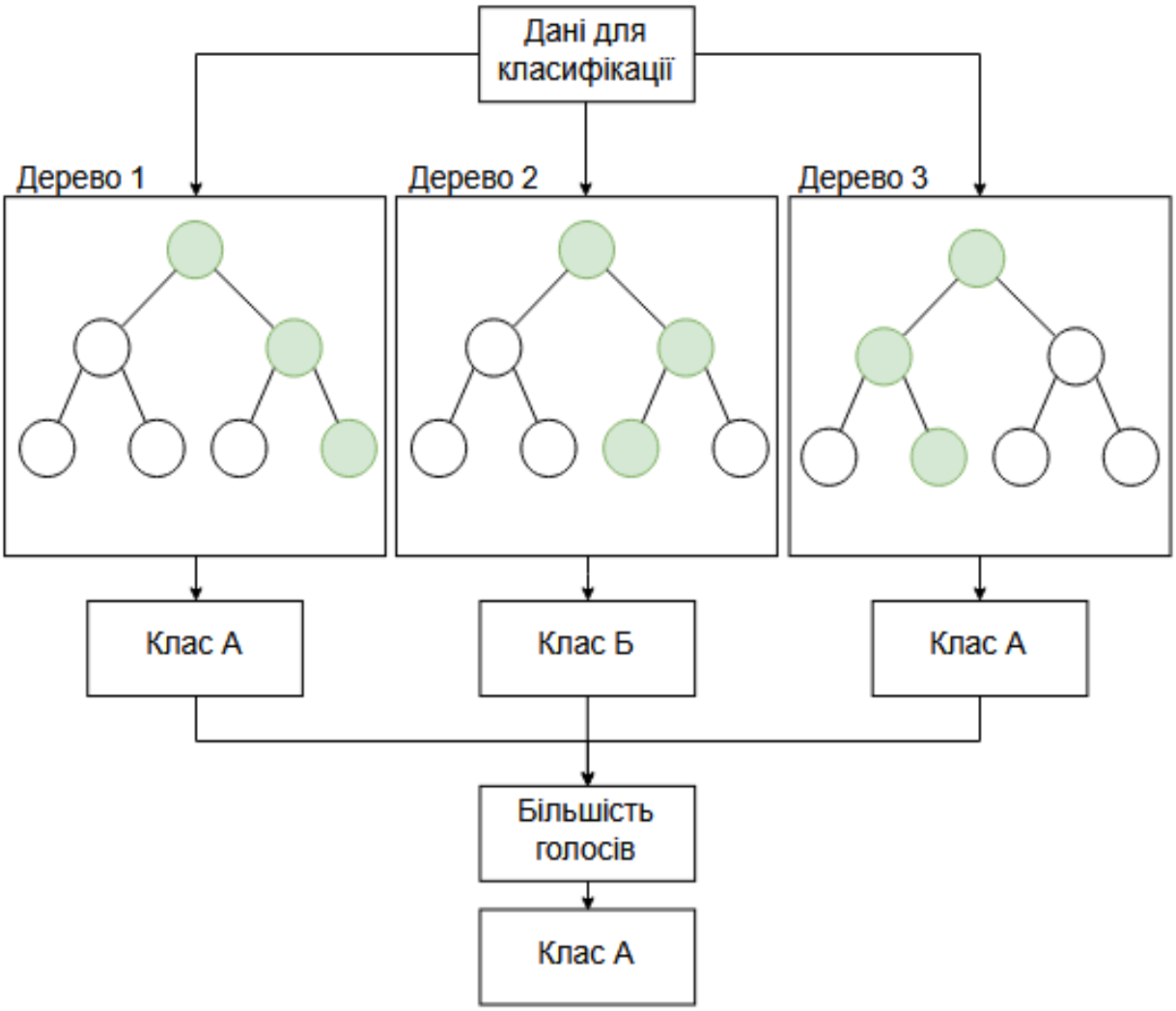


Рис. 1.5. Структура випадкового лісу

1.4. **Нейронні мережі**

Крім методів машинного навчання великої популярності здобули також нейронні мережі. Вони дуже часто використовуються для вирішення різноманітних задач, що потребують складних обчислень. До цих задач відносяться класифікація даних за ознаками, передбачення різноманітних числових даних, таких як ріст цін на бензин, а також розпізнавання різних об’єктів.Нейронні мережі являють собою програмну інтерпретацію людського мозку. Вони складаються з штучних нейронів, які послідовно з’єднані між собою синапсами, і дозволяють не лише обробляти, а і “запам’ятовувати” інформацію для подальшого її використання при обробці.

Штучні нейрони поділяються на вхідні, вихідні та приховані. Нейронні мережі складаються з вхідного шару, одного або декількох прихованих шарів, які обробляють інформацію, отриману з вхідного шару, та вихідного шару, який виводить кінцевий результат [9].

Окрім вхідного шару, вхідними даними для усіх нейронів є сума вихідних даних нейронів з попереднього шару. Просумовані дані нормалізуються (приводяться до необхідного діапазону) функцією активації. У якості функції активації може виступати лінійна функція $f(x) = x$, сигмоїд $f(x) = \frac{1}{1+e^{-x}}$, гіперболічний тангенс $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$, тощо.

Зв'язок між двома штучними нейронами називається синапсом. В кожного синапса є своя вага. При передачі даних з одного нейрону до іншого інформація змінюється, завдяки множенню на відповідну вагу.

Кожна нейронна мережа має пройти тренування. Один цикл проходження усіх тренувальних наборів даних називається епохою.

Оскільки нейронні мережі оперують числовими даними, для класифікації текстового вмісту електронних повідомлень необхідно спочатку перетворити текст на набір векторів ознак. Для цього з повідомлення видаляється все, крім тексту, після чого проводиться видалення стоп-слів, оскільки вони не впливають на зміст повідомлення. Далі повідомлення розбиваються на речення, а речення на слова, з яких створюється словник. Для кожного речення створюється вектор, значення якого – кількість повторень кожного слова зі словника в цьому реченні.

На рис. 1.6 зображені різновиди нейронних мереж. Загалом вони поділяються на мережі прямого поширення та мережі зі зворотними зв'язками.

Нейронні мережі прямого поширення передають інформацію лише в одному напрямі, а в мережах зі зворотним поширенням для того, щоб мінімізувати помилку в роботі багатошарового перцептрону дозволяють повертати на попередні шари похибку для корегування ваг.

Рис. 1.6. Схема класифікації нейронних мереж

Для класифікації повідомлень електронної пошти найдоцільніше використовувати нейронні мережі прямого поширення. Хоча нейронні мережі зі зворотними зв’язками використовуються для кластеризації, їх використання для фільтрації та класифікації текстових даних електронної пошти є надлишковим, оскільки зворотнє поширення помилки дуже ускладнює метод, не надаючи при цьому значного підвищення точності класифікації.

*1.4.1 **Одношаровий перцептрон***

Перцептрон являє собою математичну модель того, як людський мозок сприймає інформацію [9]. Класичний перцептрон складається з трьох типів елементів:

1) S-елементи. Сенсори або рецептори;
2) A-елементи. Асоціативні;
3) R-елементи. Реагуючі.

S-елементи можуть бути приймати значення 1 або 0. S-елемент зі значенням 0 знаходиться в стані спокою, а зі значенням 1 стає активним.

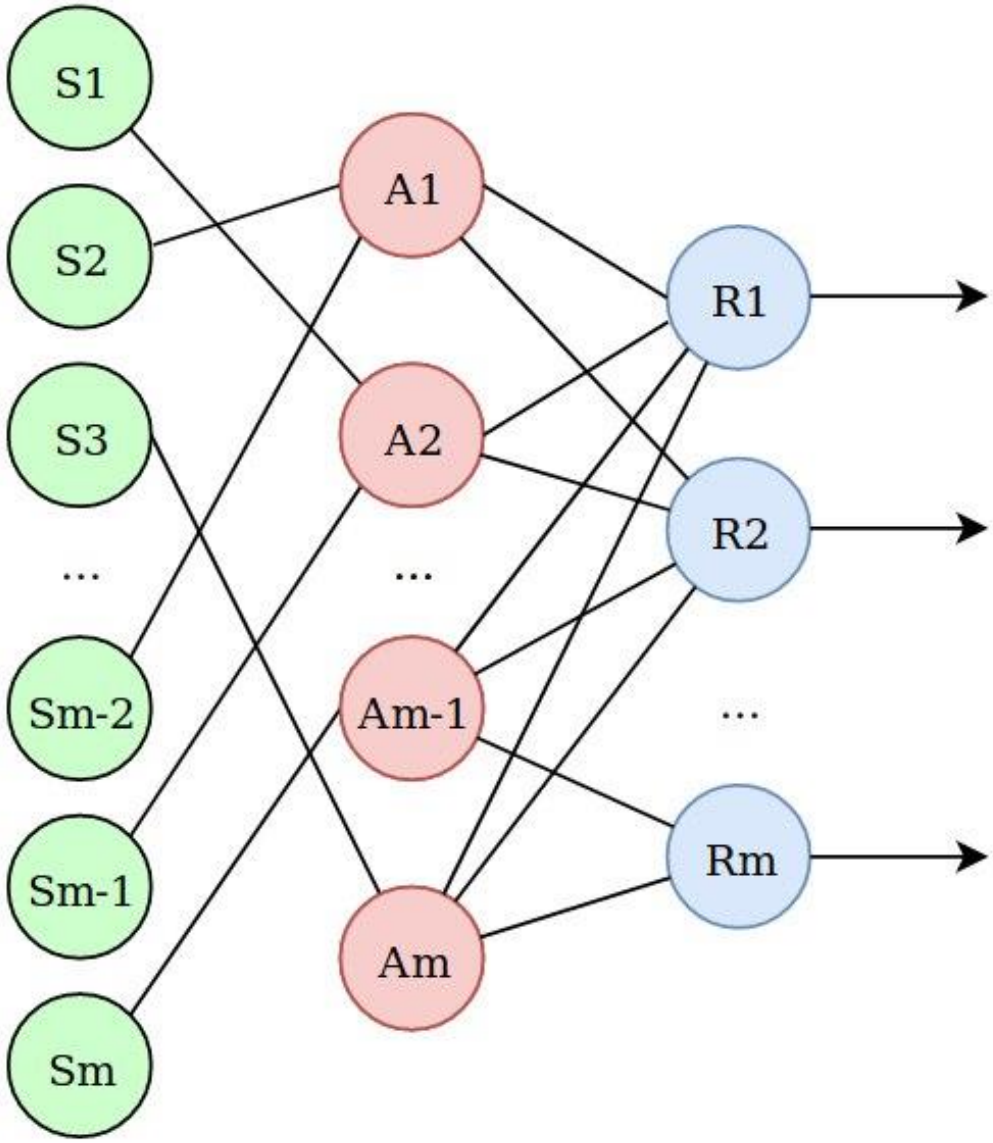


Рис. 1.7. Одношаровий перцептрон

S та A-елементи утворюють S-A зв’язки, по яким передається сигнал. Кожний з таких зв’язків має свою вагу. На рис. 1.7 зображена схема одношарового перцептрону.

Кожний S-елемент відповідає лише одному A-елементу. A-елемент може приймати сигнали від декількох S-елементів. Таким чином він накопичує значення цих сигналів. Якщо накопичене значення більше за певне граничне значення A-елемента, він активується та починає передавати сигнал зі значенням 1.

Після цього R-елемент сумує зважені сигнали A-елементів. На відміну від S-A зв’язків, значення ваг A-R зв’язків може бути будь-яким. Неактивований R-елемент передає сигнал зі значенням –1, у випадку, якщо сума зважених сигналів A-елементів перевищує заданий поріг, R-елемент активується та починає передавати сигнал зі значенням 1.

Для класифікації повідомлень електронної пошти одношаровим перцептроном, необхідно спочатку навчити мережу на тренувальних даних, а при надходженні повідомлення виконати попередню обробку. Текстова частина повідомлення повинна бути перетворена у вектори ознак, після чого можна подавати отримані вектори на входи перцептрону.

Хоча даний метод дозволяє проводити як фільтрацію, так і класифікацію текстової частини повідомлень електронної пошти за категоріями, точність його класифікації досить низька, оскільки для класифікації використовується лише один шар A-елементів.

1.4.2 ***Багатошаровий перцептрон***

Багатошаровий перцептрон за принципом аналогічний до одношарового, але на відміну від останнього, він може мати більше одного шару A-елементів [9].

Такий підхід дозволяє значно збільшити точність при класифікації текстової частини повідомлень електронної пошти. На рис. 1.8 зображена схема багатошарового перцептрону.

Одним з різновидів багатошарового перцептрону є багатошаровий перцептрон Румельхарта. На відміну від звичайного, в перцептроні Румельхарта S-A зв’язки можуть бути будь-якої ваги, та як і зв’язки A-R можуть бути навчені.

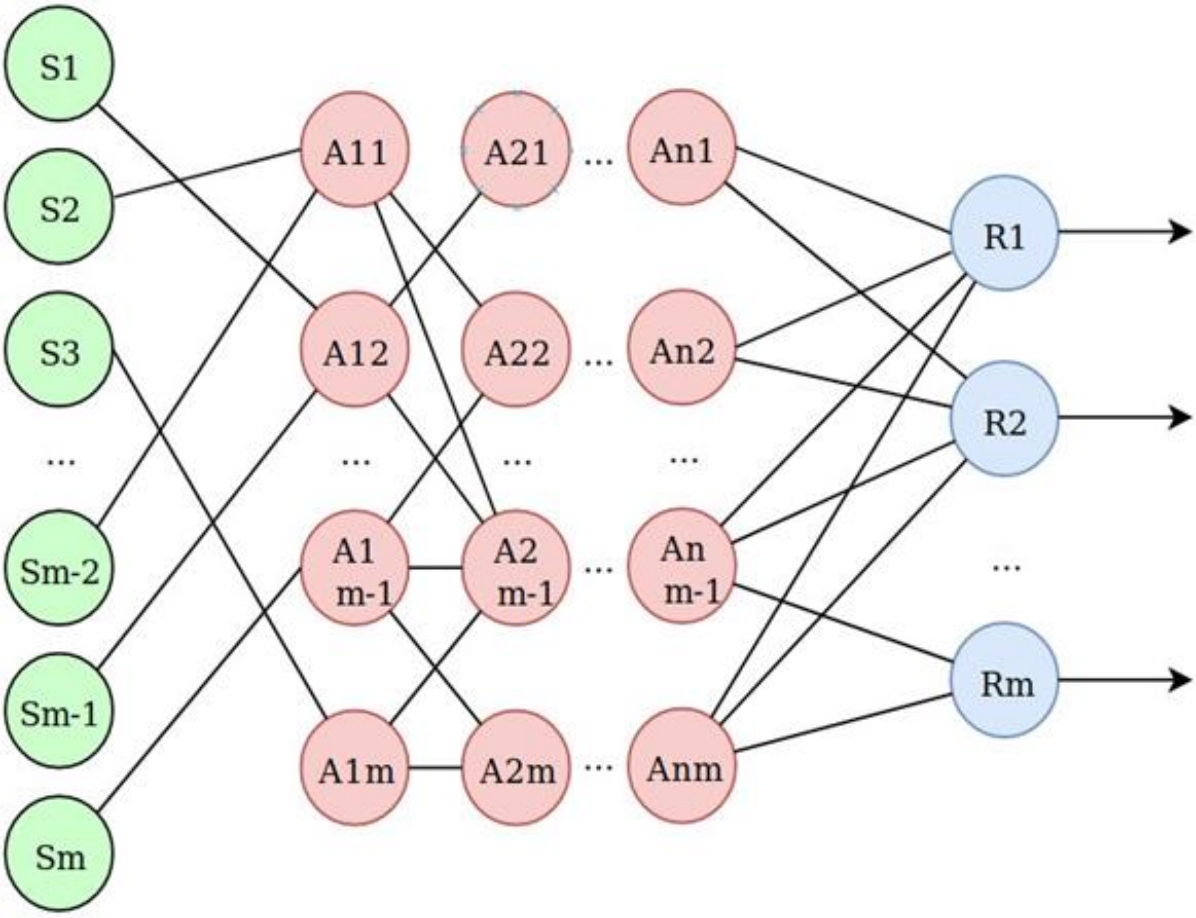


Рис. 1.8. Багатошаровий перцептрон

Перевагою перцептрона Румельхарта також є те, що навчаня відбувається за методом зворотнього поширення помилки, що підвищує точність класифікації текстової частини повідомлень електронної пошти.

Хоча точність класифікації текстових даних багатошаровим перцептроном досить висока, його використання є недоцільнім, оскільки класичні методи машинного навчання, такі як метод Баєса та метод опорних векторів, мають більшу точність та швидкість роботи, а також є більш простими в реалізації.

1.4.3 Мережа радіальних базисних функцій

Мережі радіальних базисних функцій – це такий тип штучних нейронних мереж, в яких в якості функції активації використовуються радіально-симетричні функції [9].

Радіально-симетричні функції можуть бути використані в лінійних, нелінійних, одношарових та багатошарових мережах. Мережі радіальних базисних функцій зазвичай одношарові.

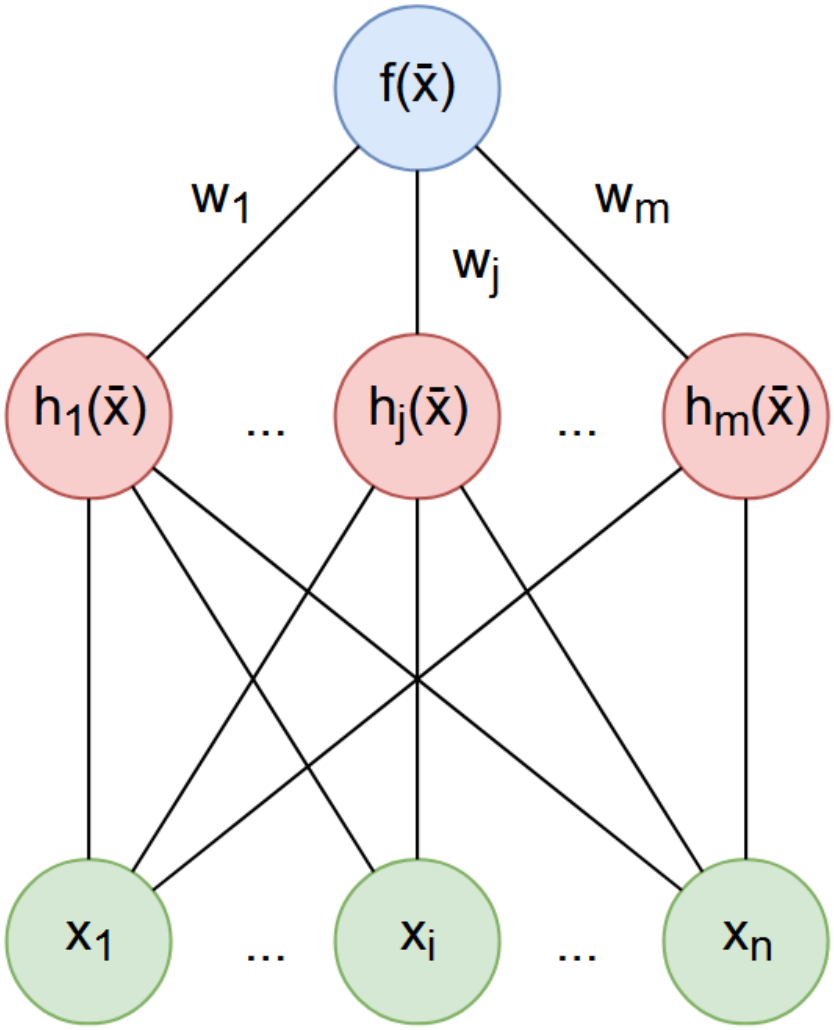


Рис. 1.9. Структура мережі радіальних базисних функцій

На вхід мережі радіальних базисних функцій подається вектор $\bar{x} = (x_1, x_2, \ldots, x_{n-1}, x_n)$. Після цього усі компоненти вектора передаються на входи базисних функцій $h_1(\bar{x})$, $h_2(\bar{x})$, …, $h_m(\bar{x})$. Далі результати усіх базисних

функцій зважуються та підраховується їх сума, після чого результат суми подається на вихід. На рис. 1.9 зображена структура одношарової мережі радіальних базисних функцій.

Одношарову мережу радіальних базисних функцій можна охарактеризувати наступною формулою:

$$f(\bar{x}) = \sum_{j=1}^{m} w_j h_j(\bar{x}) \quad (5)$$

Де x – вхідний вектор, h – базисні функції, а w – ваги базисних функцій.

Мережі радіальних базисних функцій можуть бути використані для вирішення задач класифікації, зокрема для класифікації текстового вмісту повідомлень електронної пошти.

Недоліком мереж радіальних базисних функцій, як і багатошарового перцептрону, є те що при класифікації текстових даних в порівнянні з класичними методами машинного навчання вони видають такі ж, або навіть гірші за точністю результати, маючи при цьому більш складну структуру системи, а також більший час навчання та класифікації.

1.5. **Порівняння існуючих рішень**

Кожен із запропонованих методів має свої переваги та недоліки при застосуванні для автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Методи класифікації на основі правил та на основі ваг є зовсім не придатними для вирішення поставленої задачі, оскільки мають занадто велику кількість недоліків.

Основним недоліком цих методів є те, що вони потребують постійної участі експерта у предметній області для їх регулярного оновлення. Крім того, хоча ці методи і автоматизують процес класифікації, їх точність є дуже низькою в порівнянні з класифікацією текстової частини повідомлень електронної пошти методами машинного навчання та нейронними мережами.

Методи машинного навчання мають набагато більшу точність класифікації, в порівнянні з методами класифікації на основі правил та на основі ваг.

Найефективнішим для автоматизованої фільтрації текстової частини повідомлень електронної пошти від спаму є метод опорних векторів, оскільки його точність при класифікації за двома класами є найбільшою серед запропонованих методів. Крім того, етап класифікації виконується досить швидко, що є дуже важливим фактором при класифікації у реальному часі.

Недоліком методу опорних векторів є досить низька ефективність при застосуванні для класифікації текстової частини повідомлень електронної пошти, оскільки при використанні лінійного ядра неможлива класифікація за декількома категоріями. При використанні ядер більшої розмірності значно зростає час класифікації, що є критичним при класифікації повідомлень електронної пошти, оскільки вона має відбуватись достатньо швидко, щоб користувач вчасно отримував повідомлення.

Для автоматизованої класифікації текстової частини повідомлень електронної пошти за категоріями найефективнішим є метод Баєса. Основною його перевагою є те, що він визначає ймовірність віднесення повідомлення до певної категорії. Це надає можливість відносити повідомлення до декількох категорій, або не відносити до жодної. Нажаль, цей метод не може бути використаний для одночасної фільтрації і класифікації, оскільки при такому застосуванні значно знижується точність класифікації.

Порівняльна характеристика існуючих методів для вирішення поставленої задачі наведена у табл. 1.1. Ефективність методів для автоматизованої класифікації текстового вмісту повідомлень електронної пошти оцінюється за п’ятибальною шкалою, де 1 – найгірший результат, а 5 – найкращий.

Таблиця 1.1

Порівняння існуючих методів вирішення поставленої задачі

	Фільтрація спаму	Класифікація за категоріями	Виконання в реальному часі
Метод Баєса	3	5	5
Метод опорних векторів	5	2	5
Метод k-найближчих сусідів	3	5	2
Дерева рішень	3	3	4
Випадковий ліс	3	4	4
Одношаровий перцептрон	3	3	5
Багатошаровий перцептрон	4	4	5
Мережа радіальних базисних функцій	4	4	5

Використання інших методів машинного навчання для класифікації текстової частини повідомлень електронної пошти є недоцільним, оскільки вони є менш ефективними в порівнянні з методом Баєса та методом опорних векторів. Недоліком дерев рішень та випадкового лісу є те, що повідомлення у будь-якому випадку буде віднесено до певної категорії. Хоча згаданий метод k-найближчих сусідів і надає можливість відносити повідомлення до декількох категорій, є досить повільним, оскільки потребує перерахунку релевантності кожного повідомлення після надходження нового.

Хоча нейронні мережі мають достатньо високу точність при автоматизованій класифікації текстового змісту повідомлень електронної пошти, їх застосування є надлишковим, оскільки при набагато більшій складності системі, точність їх класифікації нижча ніж при використанні методів машинного навчання, таких як метод Баєса та метод опорних векторів.

1.6. **Висновки розділу 1**

В даному розділі розглянуті існуючі методи автоматизованої класифікації текстових даних з точки зору їх застосування для фільтрації та класифікації текстового вмісту повідомлень електронної пошти.

Проведено порівняльний аналіз, в ході якого виявлено усі переваги та недоліки існуючих методів, а також визначено найкращі методи для вирішення поставленої задачі.

Найбільш ефективним класом методів для автоматизованої класифікації текстового вмісту електронних повідомлень є методи машинного навчання, оскільки вони надають достатньо високу точність та швидкість класифікації.

Виявлено, що найбільш ефективним для фільтрації електронної пошти від спаму є метод опорних векторів, а для класифікації за категоріями – метод Баєса.

Завдяки введенню порогового значення ймовірності на кроці класифікації методом Баєса можна відносити повідомлення відразу до декількох категорій.

2. КОМБІНОВАНИЙ МЕТОД АВТОМАТИЗОВАНОЇ КЛАСИФІКАЦІЇ ТЕКСТОВОГО ВМІСТУ ЕЛЕКТРОННОЇ ПОШТИ

При використанні єдиного методу для класифікації електронної пошти всі повідомлення відносяться до одного з класів, таких як “спам”, “новини”, “соціальні мережі”, “форум” тощо. Хоча таке рішення є більш ефективним з точки зору простоти реалізації та швидкості навчання, воно негативно впливає на точність.

Для підвищення точності класифікації пропонується комбінований метод автоматизованої класифікації текстового вміст, структура якого зображена на рис. 2.1.

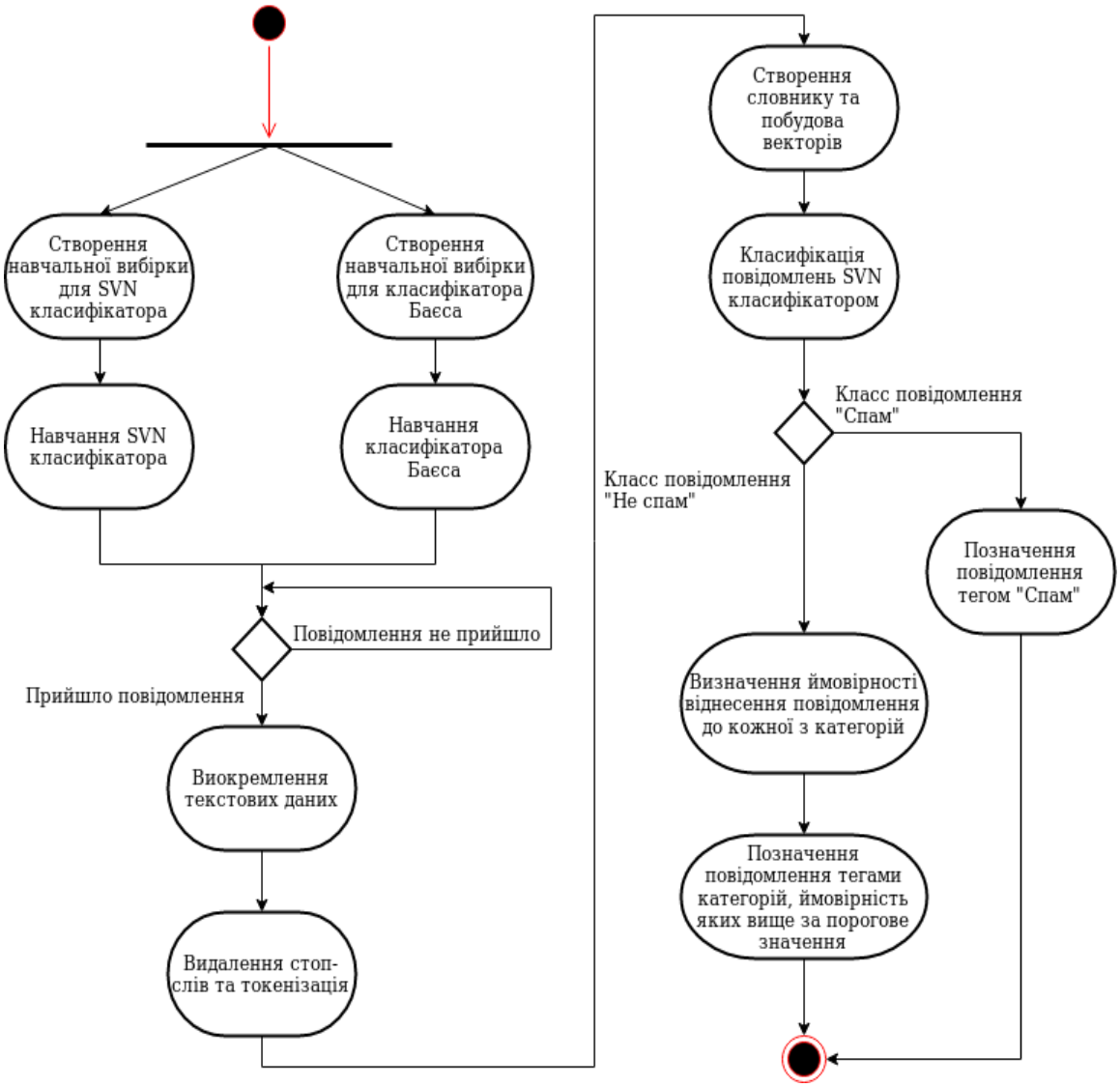


Рис. 2.1. Структура комбінованого методу класифікації повідомлень

На початку проводиться очищення текстових даних, в процесі якого з повідомлення видаляється все, крім тексту. Після цього проводиться видалення стоп-слів, оскільки вони не впливають на зміст повідомлення. Далі повідомлення розбиваються на речення, а речення на слова, з яких створюється словник. Для кожного речення створюється вектор, значення якого – кількість повторень кожного слова зі словника в цьому реченні.

Тренування обох класифікаторів відбувається окремо на сервері на незалежних даних, тому жодним чином не впливає на швидкість роботи метода для користувача.

Для фільтрації спам повідомлень обрано метод опорних векторів, оскільки його застосування з лінійним ядром є дуже ефективним для віднесення повідомлення до одного з двох класів: “спам”, “не спам”.

Для класифікації відфільтрованих повідомлень найефективніше використовувати метод Баєса, але замість віднесення повідомлення до найбільш ймовірної категорії відносити його до категорій, ймовірність яких перевищує задане порогове значення ймовірності.

2.1. **Етап тренування**

*2.1.1 **Навчальна вибірка***

В якості даних для навчальної вибірки можуть виступати як повідомлення з персональної поштової скриньки, так і набори даних, наявні в публічному доступі.

Нажаль, тренування на персональних даних зазвичай не надає бажаної точності за наступних причин:

1. Невеликий об’єм навчальної вибірки.
2. Високий рівень “забрудненості” даних. До забруднення відносяться зображення, супровідні документи, посилання, помилки у словах тощо.
3. Присутність повідомлень різними мовами. Через це зменшується вибірка повідомлень, з’являється необхідність визначати мову.

Окрім проблем, що викликають зниження точності класифікації, існує ще один важливий недолік персональної поштової скриньки в якості навчальної вибірки. За замовчанням повідомлення, що потрапляють в спам, автоматично видаляються через 30 днів. Через вибірка спам повідомлень є замалою для тренування на ній спам-класифікатора.

Набори навчальних даних, наявні в публічному доступі позбавлені недоліків персональних даних. Основною проблемою публічних наборів даних є те, що переважна більшість з них містить повідомлення англійською мовою.

Навчальна вибірка є дуже важливою для точної класифікації текстового вмісту повідомлень електронної пошти. Неправильно сформована вибірка може настільки знизити точність класифікатора, що його подальше використання буде неможливим. Найбільш якісні набори навчальних даних сформовані англійською мовою, тому саме її обрано в якості мови текстового вмісту повідомлень електронної пошти.

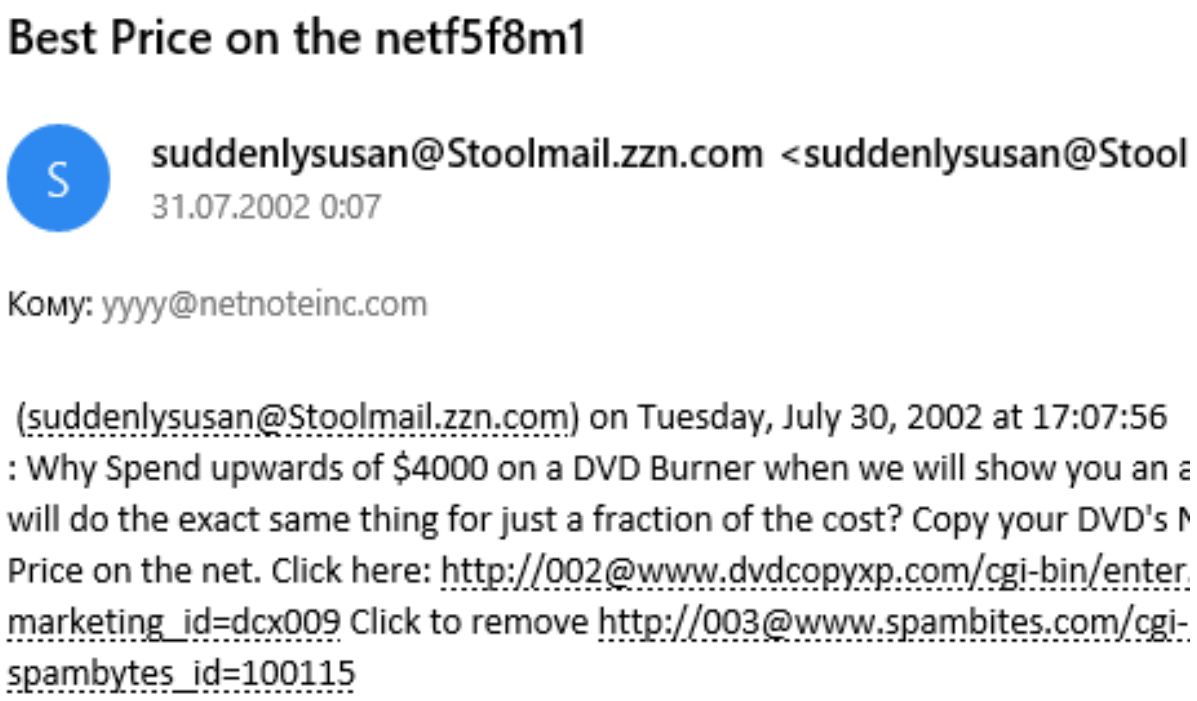


Рис. 2.2. Повідомлення з навчальної вибірки для спам-класифікатора

Для навчання двох класифікаторів необхідно дві різні навчальні вибірки, оскільки кожна з них містить повідомлення, позначені категоріями, необхідними для певного класифікатора.

В якості навчальної вибірки для спам-класифікатора обрано публічний набір повідомлень для змагання “ADCG SS14 Challenge 02 – Spam Mails Detection” на сайті www.kaggle.com. Вибірка складається з 2500 навчальних повідомлень, з яких 1721 помічені як “не спам” та 779 як “спам”, та 1827 тестових повідомлень для визначення точності класифікації після навчання. На рис. 2.2 зображений приклад повідомлення з навчальної вибірки для фільтрації спаму.

Для класифікації повідомлень за категоріями обрано публічний набір текстів новин, класифікованих за категоріями “AG news classification dataset” з ресурсу Dataturks. Набір містить 7600 текстів, взятих з сайту AgWeb.com і класифікованих за однією з чотирьох категорій: “Business”, “Sports”, “World” та “SchiTech”. На рис. 2.3 зображений приклад тексту з навчальної вибірки для класифікації за категоріями.

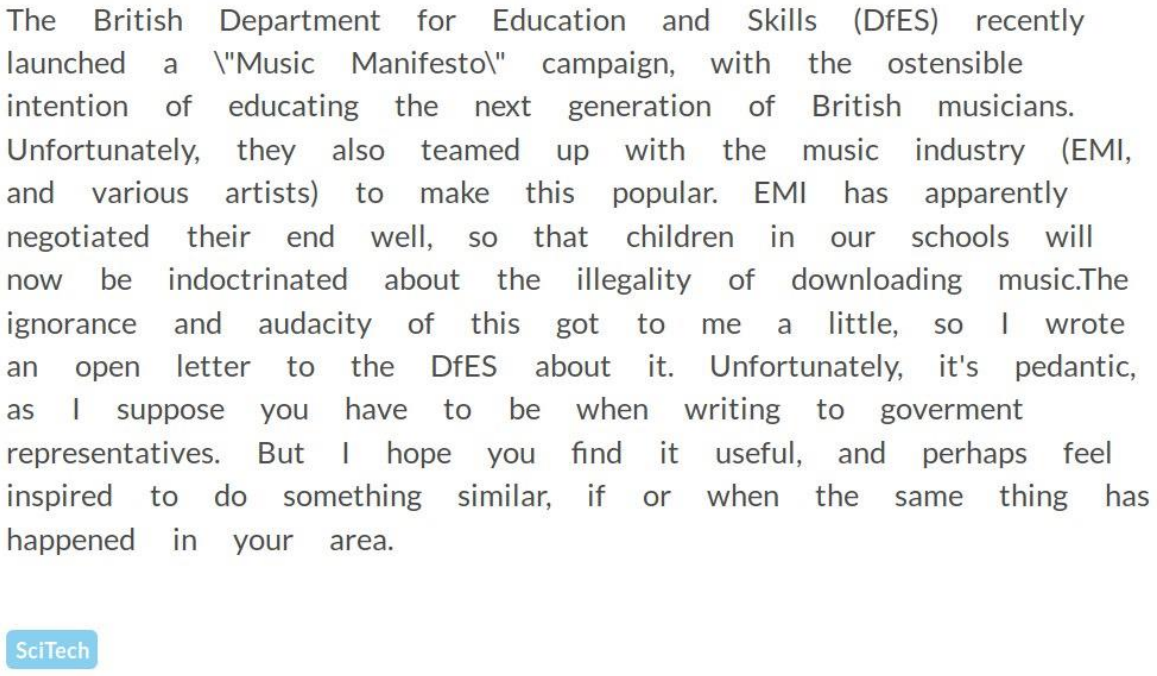


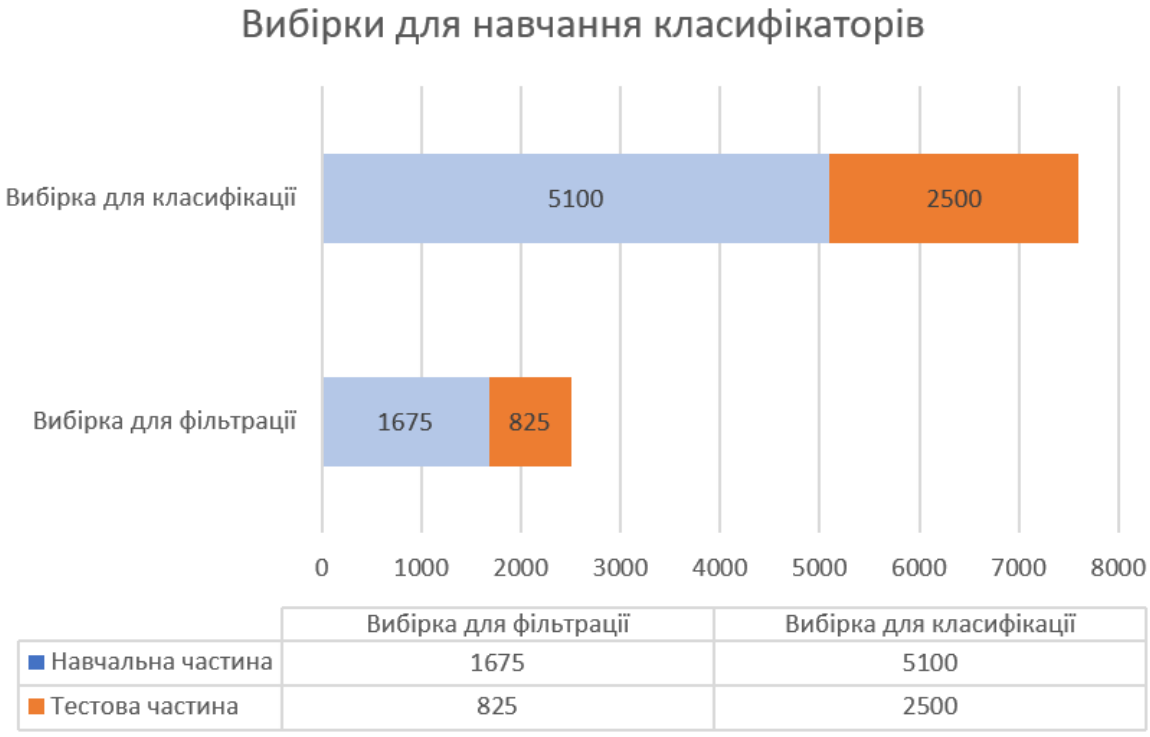
Рис. 2.3. Приклад тексту з навчальної вибірки для класифікатора за категоріями

2.1.2 Тренування класифікатора

Для тренування класифікатора необхідно розділити вибірку на навчальну та тестову частину для перевірки точності класифікації. Цей етап є важливим, оскільки надто велика кількість навчальних повідомлень призведе до перенавчання класифікатора.

Оптимальним варіантом розділення вибірки є відношення 70/30, де 70% – навчальна, а 30% – тестова частина.

На рис. 2.4 зображена діаграма розмірів навчальних вибірок для фільтрації повідомлень від спаму та класифікації за категоріями.



	Вибірка для фільтрації	Вибірка для класифікації
Навчальна частина	1675	5100
Тестова частина	825	2500

Рис. 2.4. Діаграма розмірів навчальних вибірок

Етап тренування може займати досить багато часу при використанні великих тренувальних вибірок, тому тренування обох класифікаторів відбувається один раз під час ініціалізації, щоб не впливати на час класифікації текстової частини повідомлень електронної пошти.

Прийнятним результатом тренування класифікатора на навчальній вибірці є точність його класифікації на тестовій вибірці не нижче 80%.

2.2. **Попередня обробка даних**

Після того, як виконаний етап тренування класифікаторів, можна виконувати класифікацію текстового вмісту нових повідомлень. Формат повідомлень, що надходять користувачу не підходить для автоматизованої класифікації, тому кожне нове повідомлення потребує попередньої обробки для передачі на вхід класифікатора.

На рис. 2.5 зображений приклад повідомлення, що потребує попередньої обробки.

Hi there!

Remember that you are waiting for tomorrow's **Machine Learning Meet-up!**

Day and time: August 14, 19: 30-21: 00
Place: Ring Ukraine, st. Antonovich 81 , Tower C, 8th floor.

Contacts for operational communications:
Anna 096 707 78 80 Natalia 095 272 27 33

Please inform us if your plans have changed on August 14.

See you on meeting!

Рис. 2.5. Необроблене повідомлення

*2.2.1 **Очистка повідомлення***

Зазвичай повідомлення містять не лише текстові дані, а і зображення, різноманітні супровідні документи, відео-матеріали тощо. Будь-які не текстові дані мають бути видалені з тексту, оскільки вони не можуть бути використані для класифікації.

Після видалення з повідомлення усіх об’єктів, що не відносяться до тексту, необхідно провести очистку текстових даних. З тексту видаляються усі нерелевантні символи. До них відносяться:

1) розділові знаки
2) символи переносу на наступний рядок та табуляції;
3) лапки, дужки, дефіс, нижнє підкреслення тощо.

Винятком можуть бути символи, які складають змістову одиницю в комбінації з число-буквенними символами. Прикладом такої змістової одиниці є вказаний в повідомленні час в форматі “hh:mm”, оскільки при видаленні символу двокрапки, числа, окремо один від одного, не несуть жодного змісту.

Далі виконується токенізація тексту. Текстове повідомлення розділяється на окремі слова. Іноді виконувати додаткове розділення на речення, але в контексті автоматизованої класифікації текстового вмісту повідомлень електронної пошти це є недоцільним, оскільки кінцевим результатом попередньої обробки має бути саме вектор слів, а не вектор речень.

Наступним кроком є видалення з повідомлення усіх стоп слів. Такі слова лише “забруднюють” повідомлення, не впливаючи на його зміст. В англійській мові найпопулярнішими стоп словами є “the”, “is”, “at”, “which” та “on”.

Крім цього, видаляються також усі нерелевантні слова. До таких слів можуть бути віднесені інтернет посилання на зовнішні ресурси, згадування в соціальних мережах, номера мобільних телефонів, електронні адреси тощо.

Після видалення нерелевантних слів та стоп-слів необхідно виконати приведення різноманітних форм усіх слів до словникової форми, наприклад “operational” буде приведено до форми “operation”. Крім цього потрібно виконати виправлення помилок в словах, де це можливо.

На рис. 2.6 зображене повідомлення після очистки. Жовтим кольором виділені слова, які в подальшому будуть оброблені як окремі токени.

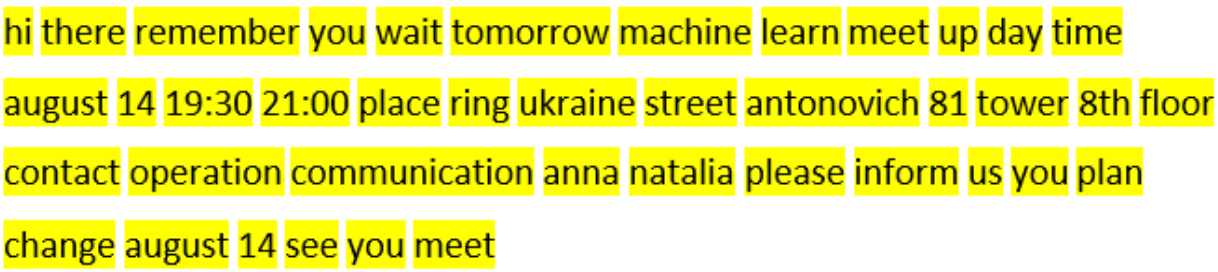


Рис. 2.6. Очищене повідомлення

2.2.2 ***Представлення даних***

Оскільки усі методи машинного навчання в якості вхідних даних приймають лише числові значення, отримане очищене та токенізоване повідомлення необхідно представити у вигляді чисел. Класичним і досить ефективним способом представлення текстових даних у вигляді вектора чисел є “мішок слів”.

Для представлення тексту у вигляді “мішка слів” необхідно створити словник усіх слів з тренувального набору даних. Кожному слову з набору буде відповідати унікальний індекс зі словника.

Кожне нове повідомлення можна виразити у вигляді вектора, довжина якого буде відповідати довжині словника, тобто кількості унікальних слів в ньому. На кожній позиції цього вектора буде зберігатися число, а саме кількість зустрічей даного слова в повідомленні. Хоча цей підхід не враховує порядок слів в реченні, він є досить ефективним при вирішенні задач класифікації тексту. На рис. 2.7 зображене векторне представлення речення з використанням мішка слів.

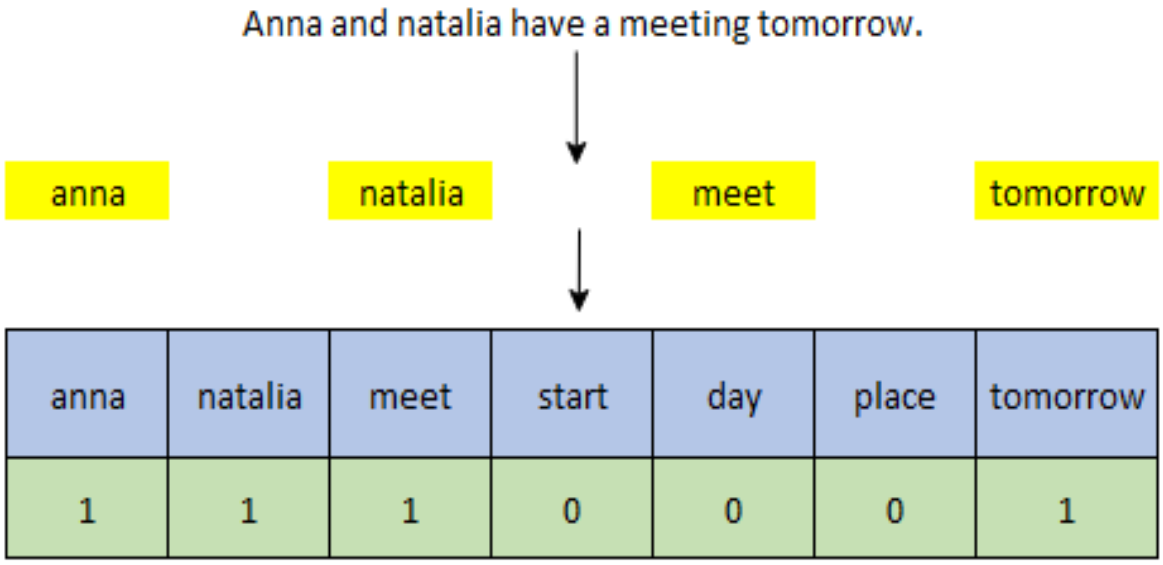


Рис. 2.7. Представлення речення у вигляді вектора

Після формування “мішка слів” та перетворення текстового вмісту повідомлення електронної пошти до векторного вигляду, його можна використовувати в якості вхідних даних для обох класифікаторів.

2.3. **Етап класифікації**

2.3.1 ***Фільтрація від спаму***

Для фільтрації від спаму обрано метод опорних векторів. За результатами аналізу, виконаного в першому розділі, цей метод, при використанні лінійного ядра, є найкращим для розділення повідомлень на два класи.

Після етапу тренування класифікатор, що базується на методі опорних векторів має гіперплощину, побудовану на основі векторів тренувальних повідомлень. Після надходження нового повідомлення і перетворення його у вектор, воно подається на вхід класифікатора.

Для класифікації нового вектора використовується класифікуюча функція $F(x)=sign((w, x)+b)$, в якій (w, x) – скалярний добуток нормального вектора до розділяючої гіперплощини та вектора, що необхідно класифікувати [5]. Якщо $F(x)=0$, повідомлення потрапляє до класу “спам”, якщо $F(x)=1$ – до класу “не спам”.

На рис. 2.8 схематично зображена класифікація вектора нового повідомлення при надходження.

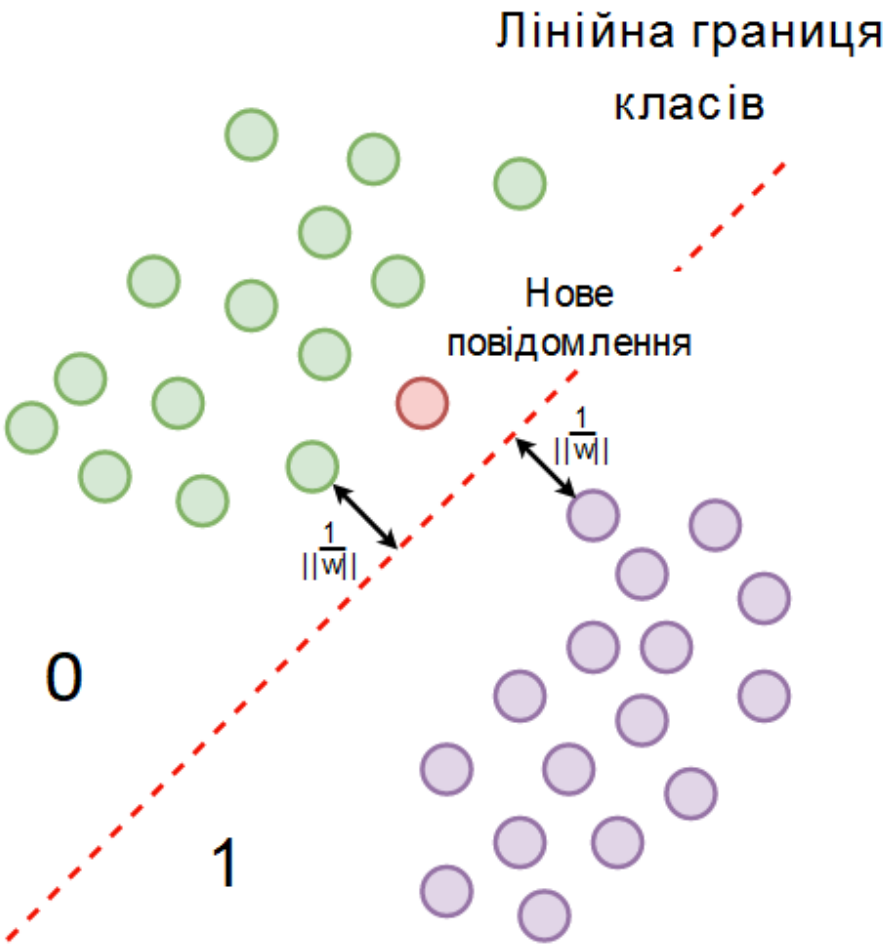


Рис. 2.8. Класифікація вектору нового повідомлення

*2.3.2 **Класифікація за категоріями***

За результатами аналізу, проведеного в першому розділі, визначено, що метод Баєса є найкращим для класифікації повідомлень за категоріями, тому саме його обрано для застосування в комбінованому методі автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Після етапу фільтрації, якщо повідомлення не потрапило до класу спам, його числовий вектор без змін потрапляє на вхід класифікатора за категоріями. В іншому випадку, класифікація припиняється на кроці фільтрації від спаму.

Щоб мати можливість відносити повідомлення не до одного, а до декількох класів, з формули (1.3) приберемо аргумент максимізації, залишивши лише частину теореми Баєса:

$$P(c|d) = P(d|c)P(c) \text{ (2.1)}$$

Оскільки за методом Баєса робиться припущення, що слова в повідомленні не залежать одне від одного, тобто поява слова не залежить від контексту [4], то умовна ймовірність віднесення повідомлення до одної з категорій буде визначатись як добуток умовних ймовірностей усіх слів *w*, з яких складається це повідомлення. Отримаємо наступну формулу ймовірності віднесення повідомлення до категорії:

$$P(d|c) = \prod_{i=1}^{n} P(w_i|c) \text{ (2.2)}$$

Для пошуку ймовірності віднесення слова до класу скористаємось наступною формулою:

$$P(w_i|c) = \frac{W_{ic}}{\sum_{k \in N} W_{kc}} \text{ (2.3)}$$

В формулі (2.3) W_{ic} – скільки разів слово w_i зустрічається в повідомленнях класу c з навчальної вибірки, W_{kc} – кількість слів в усіх повідомленнях цього класу, а N – кількість усіх слів в словнику.

Оскільки може трапитись випадок, що при класифікації в повідомленні присутні слова, яких нема в словнику, в такому випадку необхідно зробити зсув, тобто збільшити кількість кожного зі слів на 1. В такому випадку в формулі (2.3) значення чисельника збільшиться на 1, а знаменника на N:

$$P(w_i|c) = \frac{W_{ic}+1}{\sum_{k \in N} W_{kc}+N} \text{ (2.4)}$$

Після підстановки формула (2.2) матиме наступний вигляд:

$$P(d|c) = \prod_{i=1}^{n} \frac{W_{ic}+1}{\sum_{k \in N} W_{kc}+N} \text{ (2.5)}$$

Для оцінки безумовної ймовірності класу P(c) можемо скористатись відношенням кількості повідомлень з навчальної вибірки, які належать класу c, – M_c до загальної кількості повідомлень – M.

Формула буде мати наступний вигляд:

$$P(c) = \frac{M_c}{M} \text{ (2.6)}$$

З формули (2.5) випливає необхідність виконувати множення великої кількості малих чисел. Це може викликати проблему втрати точності, через певні обмеження сучасних апаратних систем. Для уникнення цієї скористаємось властивістю логарифма $\ln(xy) = \ln(x) + \ln(y)$. Застосування логарифма не змінить значення параметрів, за яких досягається максимум, а змінить лише кінцеве значення виразу, оскільки логарифм є монотонною функцією.

Модифікуємо формулу (2.1), отримаємо логарифмічну оцінку ймовірності того, що повідомлення *d* належить до класу *c*:

$$q_c = \ln\big(P(d|c)\big) + \ln\left(P(c)\right) \text{ (2.7)}$$

Підставивши формули (2.5) та (2.6) в (2.7) отримаємо:

$$q_c = \sum_{i=1}^{n} \ln\left(\frac{W_{ic}+1}{\sum_{k\in N} W_{kc}+N}\right) + \ln\left(\frac{M_c}{M}\right) \text{ (2.8)}$$

Щоб мати можливість відносити одне повідомлення до декількох категорій, необхідно ввести порогове значення ймовірності *l*. Для віднесення нового повідомлення до класу *c* необхідно, щоб виконувалась наступна нерівність:

$$l \leq P(c|d) \text{ (2.9)}$$

Через втрату точності ми не можемо скористатись формулою (2.5), але можемо використати логарифмічну оцінку з формули (2.8). Оскільки ми використовували натуральний логарифм, щоб його позбутися необхідно піднести константу *e* до степеня логарифмічної оцінки (2.8).

$$P(c|d) = \frac{e^{q_c}}{\sum_{c_k\in C} e^{q_{c_k}}} \text{ (2.10)}$$

Отримали кінцеву формулу (2.10) для визначення ймовірності того, що повідомлення *d* належить до класу *c*. Найкраще порогове значення ймовірності необхідно встанови емпіричним шляхом, змінюючи *l* в межах [0.75, 0.85] при перевірці точності класифікатора на тестовій вибірці.

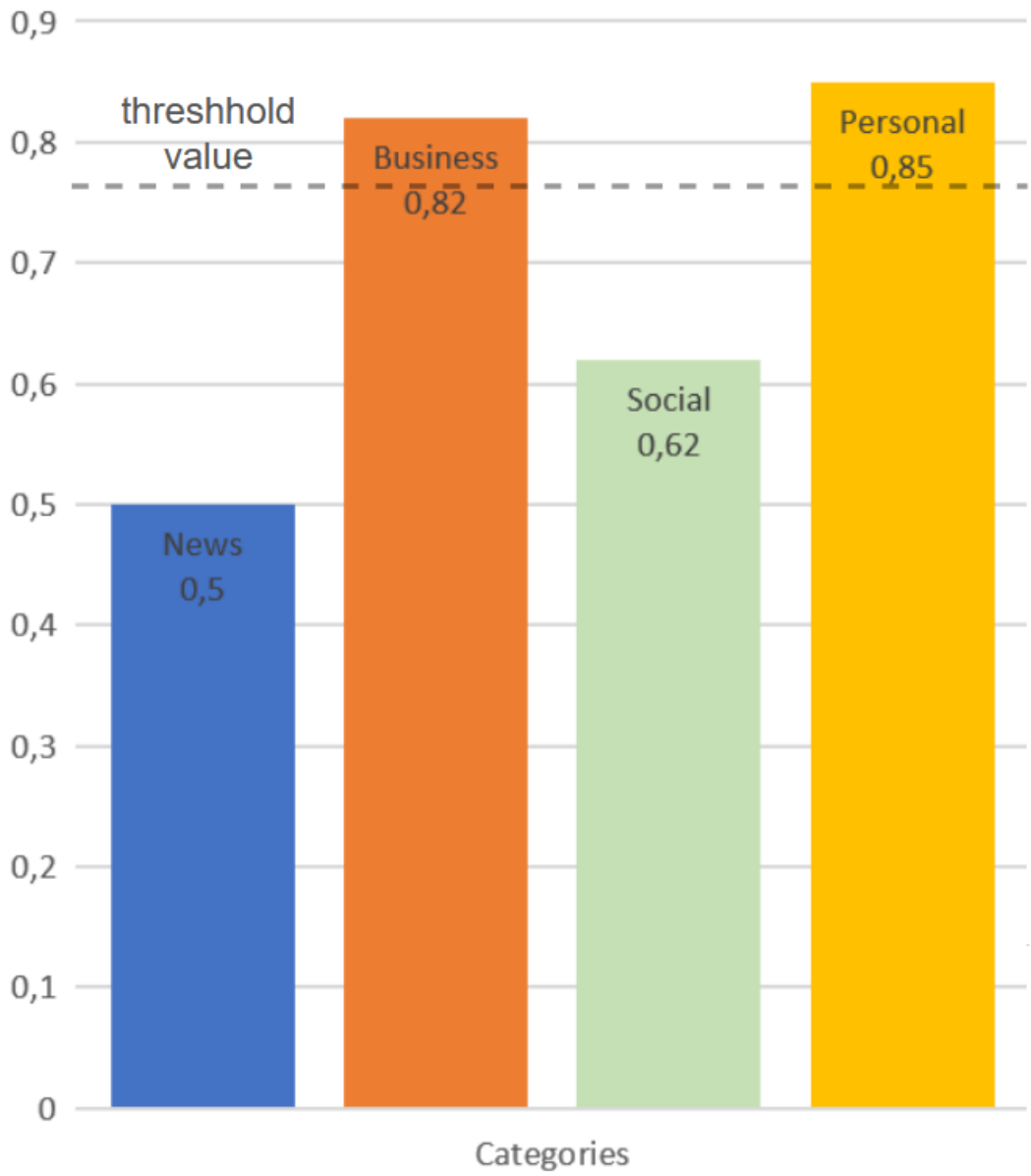


Рис. 2.9. Діаграма ймовірностей віднесення повідомлення до кожного з чотирьох класів

На рис. 2.9 зображена діаграма з результуючими ймовірностями того, що повідомлення належить до класів “News”, “Business”, “Social”, “Personal”. За допомогою порогового значення ймовірності можна визначити, що повідомлення відноситься до класів “Business” та “Personal”.

Після класифікації до класу повідомлення “не спам” додаються класи, ймовірність відношення до яких вище за порогове значення ймовірності. У випадку, якщо такі класи відсутні, повідомлення залишається не класифікованим за категорією.

2.4. **Переваги запропонованого методу**

Використання єдиного методу для фільтрації від спаму та класифікації за категоріями негативно впливає на точність, оскільки один класифікатор, натренований на вирішення обох задач набагато більше схильний до перенавчання. Завдяки розділенню етапів фільтрації текстової частини повідомлень електронної пошти від спаму та їх за категоріями значно збільшиться точність класифікації.

Очищення текстових даних, лематизація та виправлення слів з помилками також підвищують точність, так як при вирішенні задач класифікації тексту якісні вхідні дані впливають на результат більше, ніж правильне налаштування параметрів класифікатора.

Використання методу опорних векторів на етапі фільтрації повідомлень електронної пошти дозволяє не проводити зайву класифікацію за категоріями у випадку, якщо повідомлення є спамом.

Модифікація методу Баєса завдяки використанню логарифмічної оцінки ймовірності знімає проблему втрати точності, характерну для сучасних апаратних систем. Крім цього, введення порогового значення ймовірності замість аргумента максимізації надає можливість відносити повідомлення до декількох категорій, або не відносити до жодної.

Запропоновані модифікації комбінованого методу автоматизованої класифікації текстового вмісту електронних повідомлень надають можливість підвищити точність класифікації, в порівнянні з існуючими методи вирішення поставленої задачі.

2.5. **Висновки до розділу 2**

В цьому розділі запропоновано та описано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Описано основні етапи методу, а саме: підбір наборів навчальних даних, тренування класифікаторів, очищення текстового вмісту

повідомлення, створення словника, представлення текстового вмісту повідомлення у вигляді вектору, фільтрація повідомлень від спаму та класифікація відфільтрованих повідомлень.

В ході опису етапів класифікації запропоновано використовувати логарифмічну оцінку ймовірності для запобігання втрати точності, а також порогове значення ймовірності для одночасного віднесення повідомлення до декількох категорій.

До переваг комбінованого методу можна віднести підвищену точність класифікації текстового вмісту повідомлень електронної пошти, в порівнянні з існуючими методами, а також можливість класифікувати повідомлення за декількома категоріями, або не класифікувати взагалі.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Вибір технологій для вирішення поставленої задачі

3.1.1 Технології серверної частини

В якості мови програмування серверної частини обрано Python 3.7. Це високорівнева мова програмування, розроблена в 1991 році Гвідо ван Россумом [10].

Основною перевагою мови Python є велика кількість різноманітних бібліотек. Python є однією з найпопулярніших мов програмування для вирішення задач машинного навчання, завдяки чому створено безліч бібліотек з інструментами, які полегшують розробку програм для роботи з даними, зокрема для їх класифікації. До найпопулярніших бібліотек для роботи з даними відносяться Pandas, Tensorflow, Scikit-learn, NLTK.

Крім цього, мова Python є дуже потужним інструментом для розробки серверної частини веб-додатків. Завдяки таким фреймворкам, як Django, Flask, Pyramid тощо можна розробляти як монолітні додатки, так і додатки, що базуються на мікросервісній архітектурі. Існуючі бібліотеки дають можливість обирати між вже визначеною структурою системи, яка забезпечує легкість розробки і стабільність в роботі, та можливістю формувати структуру розробнику, що надає системі гнучкість та легку масштабованість.

Також значною перевагою є кросплатформеність, динамічна типізація та синтаксис мови Python. Завдяки спрощенню багатьох синтаксичних конструкцій програма на Python є легшою для розуміння, в порівнянні з іншими мовами програмування. Це дає можливість зосередитись на розробці логіки програми, завдяки чому підвищується ефективність та швидкість розробки.

В якості веб-фреймворка для розробки був обраний Flask. Цей мікрофреймворк надає можливість розроблювати більш гнучкі інтерфейси та вносить більше контролю в процес розробки. Крім того, Flask надає зручний інтерфейс для роботи з базою даних та сесіями користувача [11].

Для обробки та представлення даних обрано фреймворк Pandas. Він дозволяє автоматизувати процес оформлення даних для передачі на вхід класифікаторів, а також збільшити швидкість їх обробки завдяки формуванню датафреймів. Структура датафрейма надає можливість також легко проводити аналіз даних, такий як пошук медіани, середнього арифметичного, моди тощо.

В якості фреймворку для створення та тренування класифікаторів, а також визначення точності моделей обрано Scikit-learn. Цей фреймворк надає потужний інтерфейс для створення класифікаторів, що базуються на різних методах машинного навчання. Крім того, Scikit-learn надає можливість проводити векторизацію тексту, розділяти навчальну вибірку на тренувальну та тестову частини, отримувати інформацію про хід процесу тренування класифікатора, точність натренованої моделі класифікатора та важливість ключових ознак.

Оскільки тренування класифікатора спаму, який базується на методі опорних векторів, може займати від декількох хвилин на невеликих вибірках до декількох годин на вибірках з декількох десятків тисяч повідомлень, при вимкненні сервера, його перезавантаження буде займати досить багато часу. Щоб запобігти цьому, необхідно зберігати натреновану модель для її швидкого завантаження з файлу при наступному запуску сервера. Для збереження моделей обрано бібліотеку Pickle. Вона надає можливість проводити серіалізацію та десеріалізацію будь-яких об’єктів. Це дозволить зберігати не лише моделі, а і векторні представлення усіх повідомлень з навчальної вибірки.

Для очищення текстових даних, отриманих з повідомлення обрано бібліотеку Natural Language Toolkit. Вона дозволяє виконувати розділення текстового вмісту на повідомлення та слова, а також проводити лематизацію слів [8]. Крім того, в Natural Language Toolkit присутні списки стоп-слів на різних мовах, що дає можливість розширювати систему не змінюючи логіку обробки тексту.

Електронні поштові системи передають досить великі об’єми різноманітних даних. Для підтримки швидкості роботи цих систем дані необхідно швидко обробляти, зберігати та отримувати. В якості системи керування базою даних розглянуто SQLite, MySQL та PostgreSQL. SQLite та MySQL виявились досить ефективними для обробки невеликих та середніх об’ємів текстових даних. На великих наборах даних найефективнішою в плані швидкості обробки запитів виявилась PostgreSQL , тому саме її обрано в якості системи керування базою даних для зберігання повідомлень та даних користувача.

Для взаємодії серверної частини з базою даних обрано бібліотеку SQLAlchemy. Вона надає можливість створювати об’єктно-реляційні відображення для опису структури бази даних, а також доступу до цих даних без використання мови SQL. Бібліотека SQLAlchemy також дозволяє формувати моделі зі зв’язками багато-до-багатьох, уникаючи створення проміжних моделей, завдяки чому значно спрощується структура об’єктно-реляційного відображення.

Хоча прості запити до бази даних оброблюються досить швидко, ця швидкість не завжди є достатньою, через що деякі дані недоцільно зберігати в SQL базі. До таких даних відноситься інформація про поточну сессію, а також тимчасові коди. Для більш швидкої передачі цих даних їх необхідно зберігати в оперативну пам’ять, в якості такої системи обрано сховище даних Redis, яке дозволяє зберігати та отримувати інформацію по ключу максимально швидко.

В якості бібліотеки для валідації даних, отриманих з клієнтської частини, обрано Marshmallow. Ця бібліотека дозволяє формувати схеми за якими необхідно перевіряти дані формата json, отримані з запиту, що дозволяє запобігти передачі некоректних даних на рівні відображень. Крім того, при перевірці Marshmallow автоматично проводить конвертацію рядків в необхідні типи, завдяки чому максимально спрощується етап обробки запитів, що надходять до сервера.

Деякі задачі мають відпрацьовувати незалежно від падінь серверної частини. Однією з таких задач є відправлення повідомлення з підтвердженням електронної пошти користувачу при його реєстрації. Для відокремлення задач в окрему асинхронну чергу виконання використано Celery Task Queue, а саме її реалізацію в якості бібліотеки Celery на мові Python. Celery працює окремо, отримуючи задачі від сервера, які стають в чергу на виконання. Це дозволяє гарантувати виконання певної задачі, навіть при тимчасовій відмові сервера. Важливою умовою є те, що після відмови необхідно, щоб сервер знову почав працювати, оскільки Celery не виконує надіслані задачі, а лише ставить їх у чергу, яка продовжує існувати у випадку несправності сервера. На рис. 3.1 зображена схема роботи Celery Task Queue.

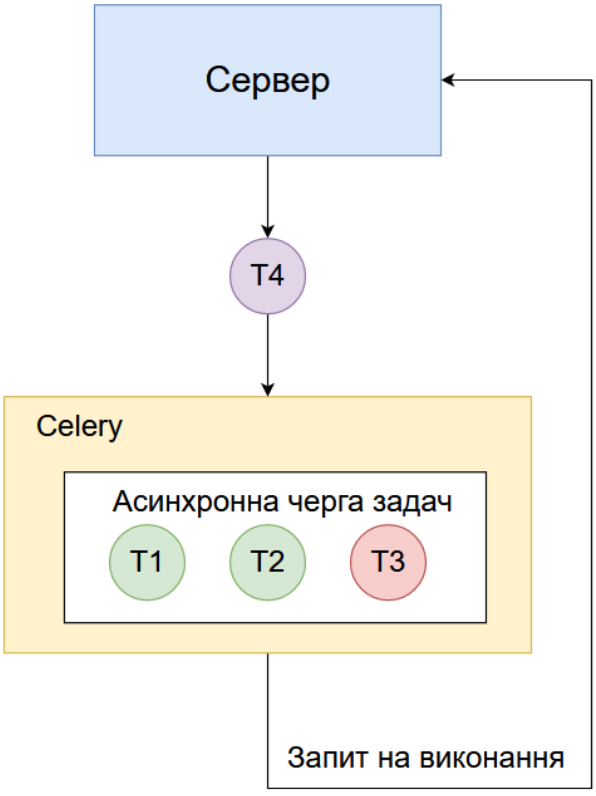


Рис. 3.1. Схема роботи Celery Task Queue

Для можливості аутентифікації користувача за допомогою соціальних мереж Google та Facebook, були використані бібліотеки з відповідними

назвами. Ці бібліотеки надаються інтерфейс для отримання детальних даних про користувачів, таких як ім’я, прізвище, зображення аватару, список повідомлень тощо.

3.1.2 Технології клієнтської частини

В якості мови програмування клієнтської частини обрано TypeScript, яку було створено та представлено компанією Microsoft в 2012 році. Ця мова розширює можливості мови програмування JavaScript, а також враховує її недоліки. Крім того, TypeScript підтримує зворотню сумісність з JavaScript [12].

Основним фреймворком для розробки клієнтської частини обрано Angular 8. Основною його перевагою є використання мови TypeScript, а також структурованість. Кожен додаток в Angular складається з компонентів та сервісів [13]. Будь-який з компонентів може бути повторно використаний на різних сторінках додатку, завдяки чому уникається дублювання коду та покращується загальна структура. В сервіси винесено логіку обробки даних додатку, крім того вони забезпечують передачу даних до компонентів, відправлення запитів до серверної частини та отримання відповідей, а також зв’язок між компонентами. Така визначеність структури додатку значно полегшує процес його розробки.

Для аутентифікації користувача за допомогою соціальних мереж Google та Facebook на клієнтській частині використано бібліотеку angularx-social-login. Вона надає спільний інтерфейс для різних соціальних мереж, використовуючи ключ додатку відповідної соціальної мережі. Бібліотека angularx-social-login може надавати базові дані користувача соціальної мережі, такі як токен для аутентифікації, ім’я, прізвище, зображення аватару тощо.

В якості бібліотеки для розробки інтерфейсу користувача обрано Angular Material UI. Ця бібліотека розроблена та оптимізована для використання з фреймворком Angular, завдяки чому є дуже легкою та

ефективною у використанні. Angular Material UI надає можливість використовувати різноманітні готові компоненти інтерфейсу у стилі Material Design [14] . Це спрощує розробку зовнішнього вигляду, а також формує єдиний стиль системи.

3.1.3 Зовнішні інтерфейси

Для реалізації можливості взаємодії клієнтської та серверної частини з соціальними мережами Google та Facebook були використані відповідні сервіси для розробників, а саме GoogleAPIs та Facebook for developers.

GoogleAPIs надає можливість створити додаток, який зможе отримувати доступ до даних різних сервісів Google, таких як Google Maps, YouTube Analytics, Google Drive тощо. На рис. 3.2. зображено інтерфейс облікових даних GoogleAPIs. Ключі API надають можливість отримати доступ до сервісу зовнішнім системам, а ідентифікатори клієнтів OAuth 2.0 - отримати інформацію про певний клієнт.

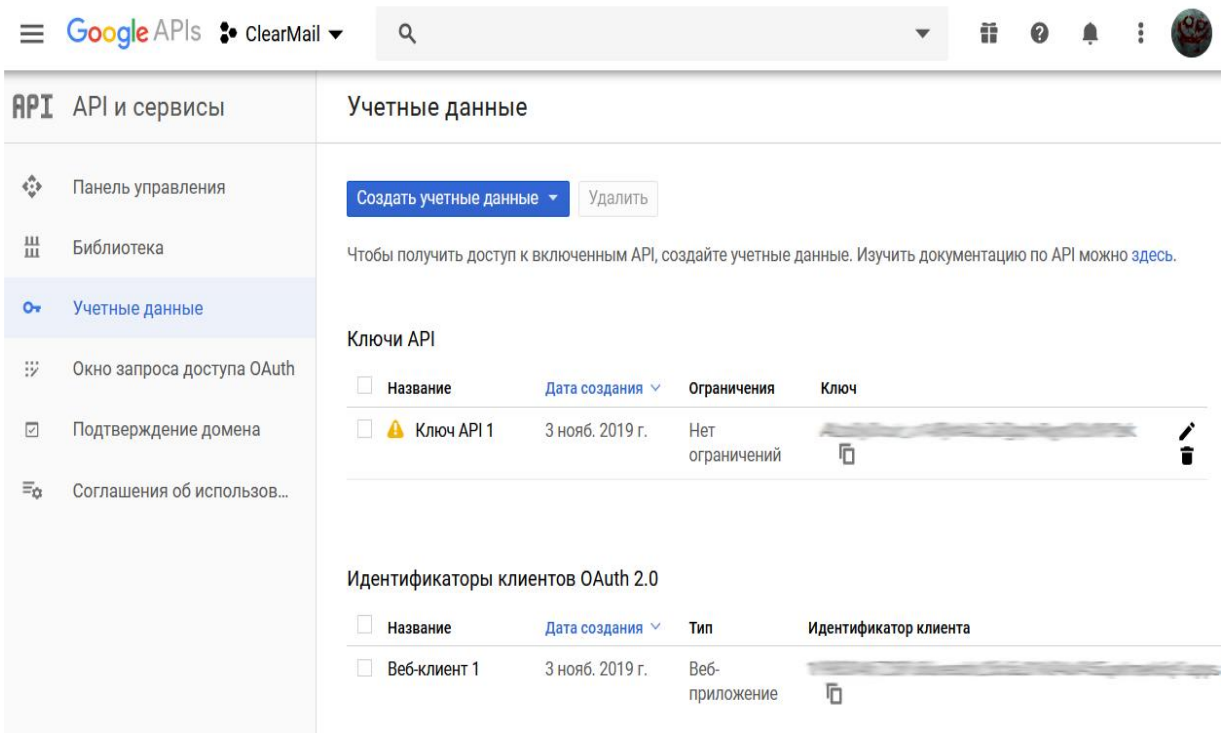


Рис. 3.2. Інтерфейс облікових даних додатку в GoogleAPIs

Facebook for developers, як і GoogleAPIs надає можливість створювати додаток та додавати до нього різні сервіси. Базова інформація, яку розробник може отримати з цих сервісів є безкоштовною, але основна їх функціональність платна. В додатку Facebook for developers можна також налаштовувати рівні доступу до додатку. В режимі розробки доступ до додатку в Facebook for developers має лише розробник. Доступ здійснюється по токену App ID. На рис. 3.3 зображено інтерфейс додатку в Facebook for developers.

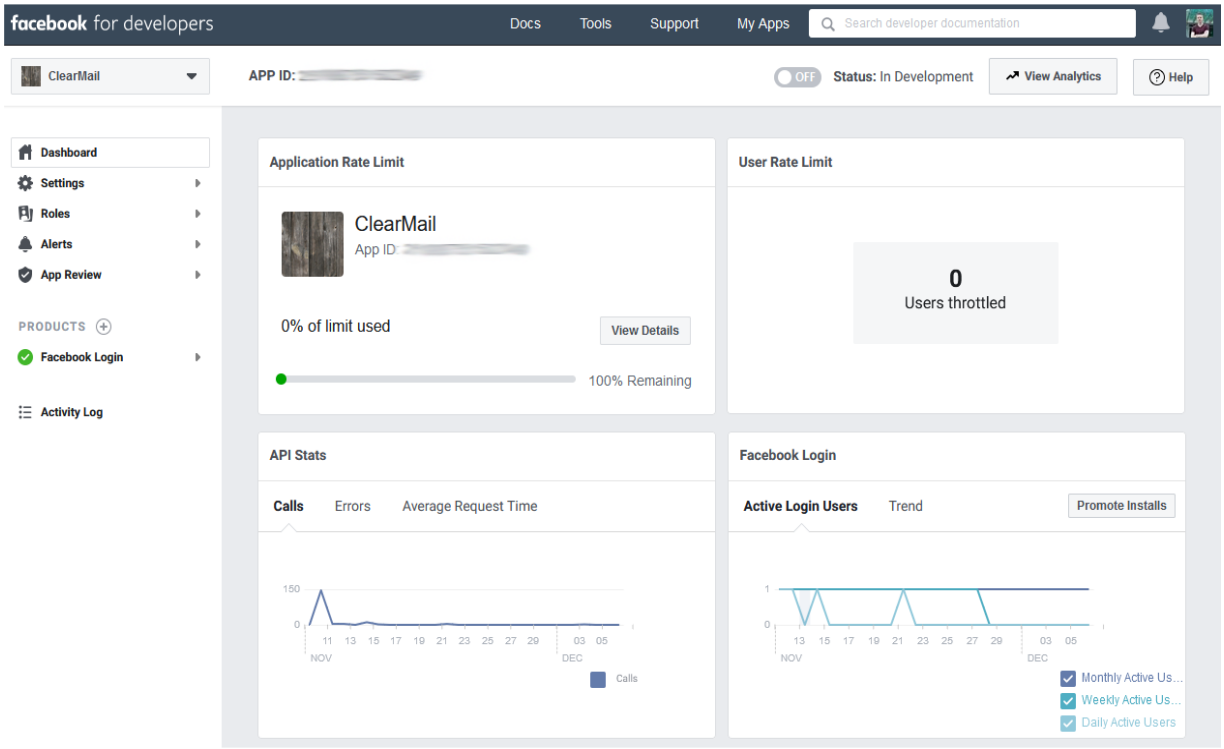


Рис. 3.3. Інтерфейс додатку в Facebook for developers

3.2. **Архітектура розробленої системи**

Для реалізації комбінованого методу автоматизованої класифікації текстового вмісту обрано створення електронної поштової системи у вигляді веб-додатку, оскільки існуючи системи не надають інтерфейсу для їх модифікації.

Розроблений веб-додаток складається з клієнтської та серверної частини. Клієнтська частина складається з компонентів та сервісів, які

забезпечують зв’язок між компонентами. Серверна частина, в свою чергу, складається з аплікацій, які взаємодіють між собою, а також з базою даних. Крім того, клієнтська та серверна частини отримують дані з ресурсів Facebook та Google через зовнішні інтерфейси.

Окремо від серверної частини працює Celery Task Queue, яка ставить в асинхронну чергу на виконання різні задачі. Загальна структура системи зображена на рис. 3.4.

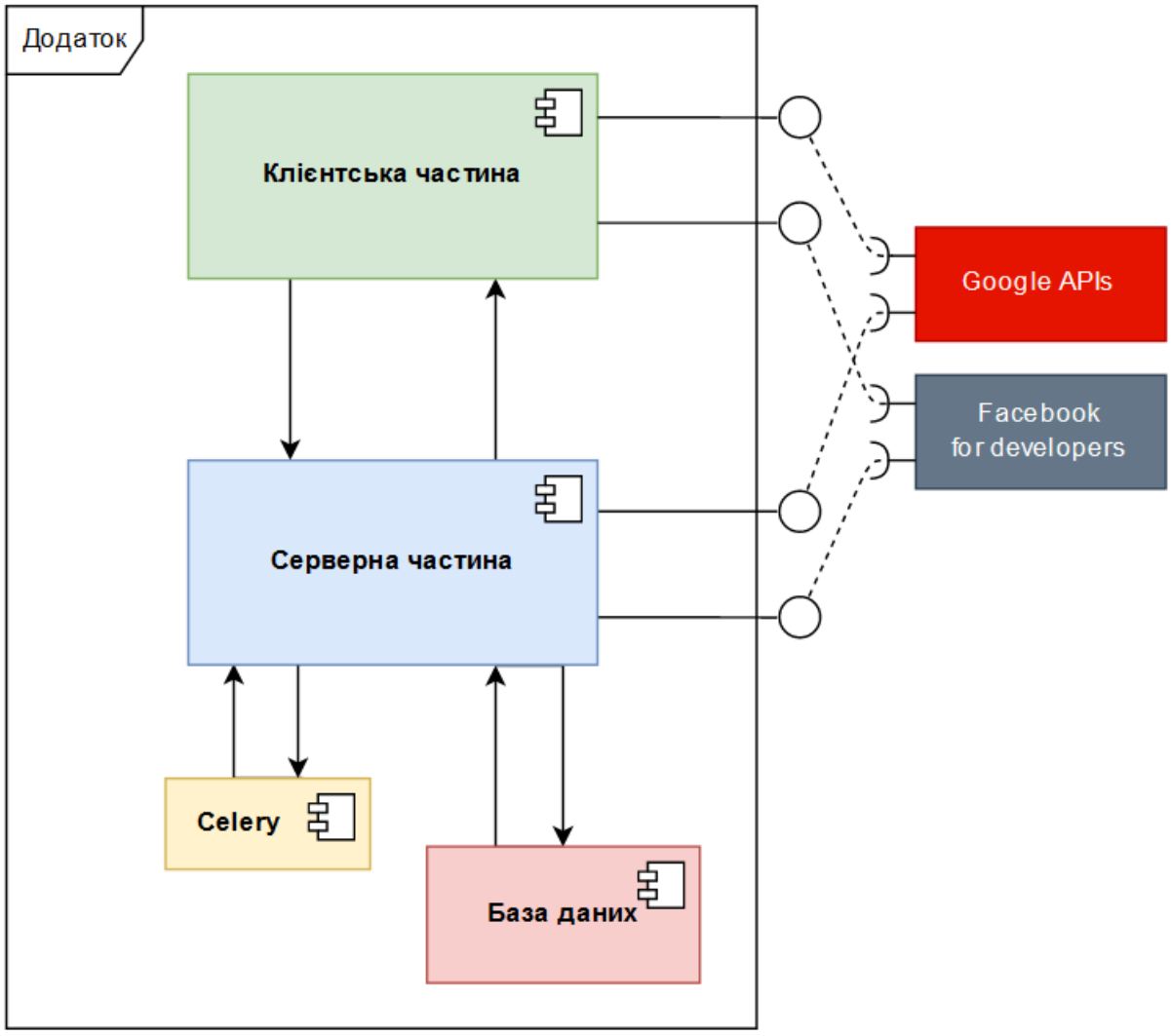


Рис. 3.4. Загальна структура веб-додатку

*3.2.1 **Серверна частина***

Серверна частина веб-додатку розроблена за шаблоном VCR (View Controller Repository). Вона складається з головної аплікації, яка в свою

чергу ініціалізує окремі модулі. Кожен з модулей являє собою об’єкт типу blueprint фреймворку Flask і реалізує інтерфейс для отримання, обробки, зберігання та відправлення даних до клієнтської частини.

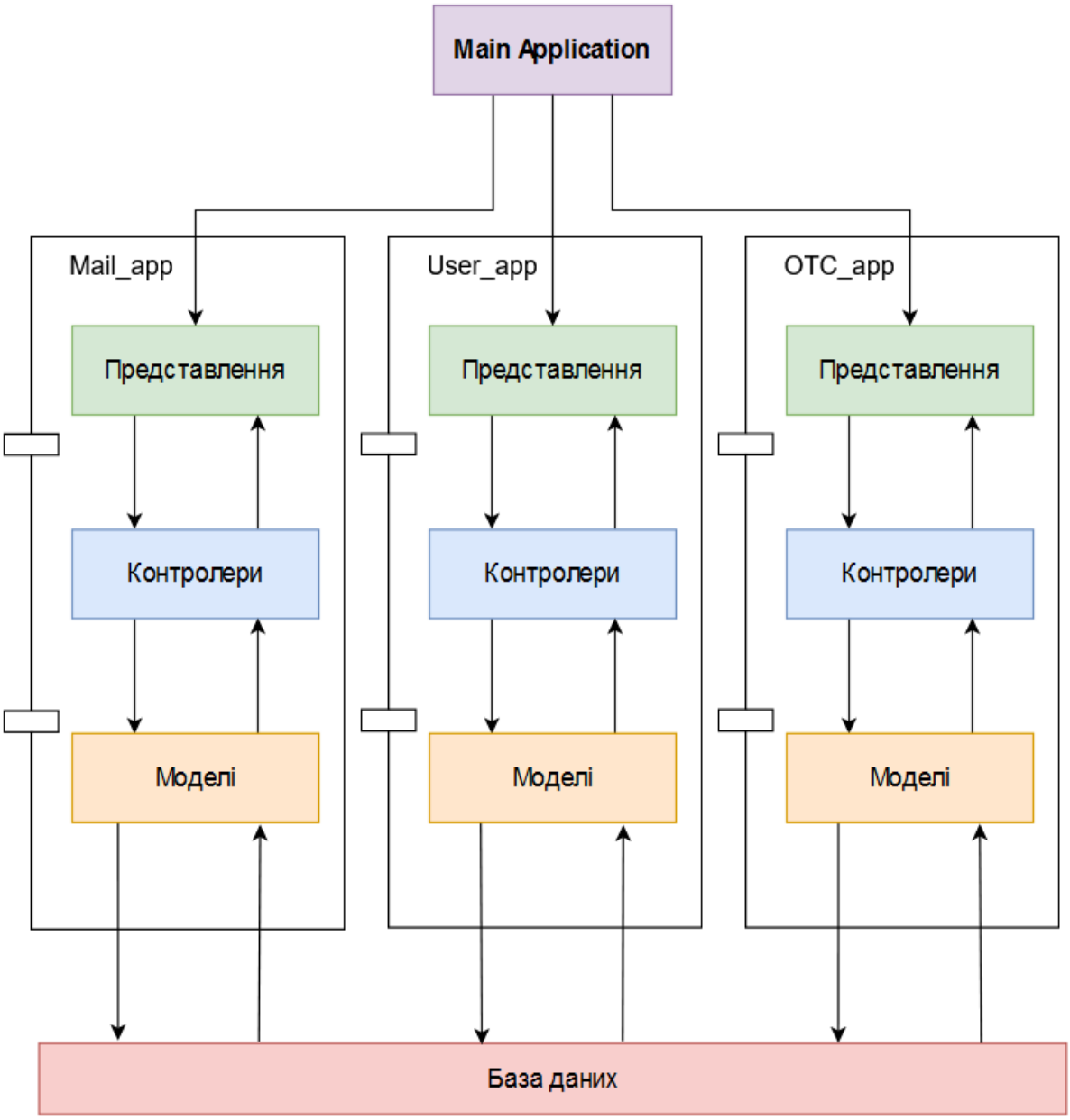


Рис. 3.5. Структура серверної частини веб-додатку

Кожен з модулів складається з трьох шарів: шара представлення (View), шара контролерів (Controller), а також шара моделей (Model/Repository). На рис. 3.5 зображена загальна структура серверної частини веб-додатку.

В розробленому додатку реалізовані наступні модулі:

1. Mail_app – модуль оброки даних текстових повідомлень. Відповідає за збереження, передачу, фільтрацію та класифікацію текстових повідомлень.
2. User_app – модуль обробки даних користувача. Відповідає за створення, аутентифікацію, активацію та редагування інформації користувача.
3. OTC_app – модуль обробки тимчасових кодів. Відповідає за генерацію тимчасових кодів для активації зареєстрованого користувача. Шар представлень відповідає за отримання запитів з клієнтської частини, валідацію отриманих даних, виклик відповідних контролерів та відправку відповідей з даними на клієнтську частину.

Шар представлень відповідає за отримання запитів з клієнтської частини, валідацію отриманих даних, виклик відповідних контролерів та відправку відповідей з даними на клієнтську частину.

Шар контролерів відповідає за обробку отриманої з представлення інформації. Контролери можуть звертатись до моделей для збереження та запиту інформації. Крім того, контролери можуть викликати один одного навіть якщо вони знаходяться в різних модулях. Таким чином реалізована взаємодія між різними модулями в додатку. Після обробки інформації контролер повертає до представлення відповідь з даними та HTTP статус кодом операції.

Шар моделей відповідає за взаємодію з базою даних. Кожна модель зі своїми атрибутами є відображенням таблиці в бази. Отримуючи запити від контролера, моделі можуть додавати, видаляти, оновлювати або повертати необхідну інформацію з бази даних.

Повноцінну роботу основних модулів забезпечує пакет допоміжних класів. В нього входять усі класи, що не відносяться до шаблону VCR, але до яких мають звертатися основні модулі для обробки даних.

Основним в пакеті допоміжних класів є модуль класифікації повідомлень, оскільки в ньому реалізовано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Модуль класифікації повідомлень складається наступних класів:

1. DatasetPreProcessor – клас обробки текстових даних, в якому реалізована логіка очищення сирих повідомлень. Кожне повідомлення проходить видалення зайвих сутностей, розділення на слова, видалення стоп-слів та лематизацію. Цей клас також оброблює всі навчальні набори даних, зберігаючи їх у придатний для класифікатора формат.
2. BaseClassifier – основний клас, в якому реалізована логіка векторизації оброблених повідомлень, тренування, розділення навчальних вибірок, перевірка точності класифікатора та зберігання моделей для подальшого використання.
3. SpamClassifier – успадковується від BaseClassifier. Реалізує класифікатор, що базується на методі опорних векторів для фільтрації повідомлень від спаму.
4. CategoriesClassifier – успадковується від BaseClassifier. Реалізує класифікатор, що базується на методі Баєса для класифікації повідомлень за категоріями.

На рис. 3.6 зображена схема діаграми класів модуля класифікації повідомлень.

Для економії часу при перезавантаженні сервера натреновані моделі спам класифікатора та класифікатора за категоріями, а також векторизатори текстового вмісту навчальної вибірки зберігаються у форматі sav, оскільки час завантаження файлу значно менший ніж повторне тренування.

Векторизатор використовується для формування “мішка слів” та перетворення усіх повідомлень з навчальної вибірки в числові вектори та подальшого розділення на тренувальну та тестову частину.

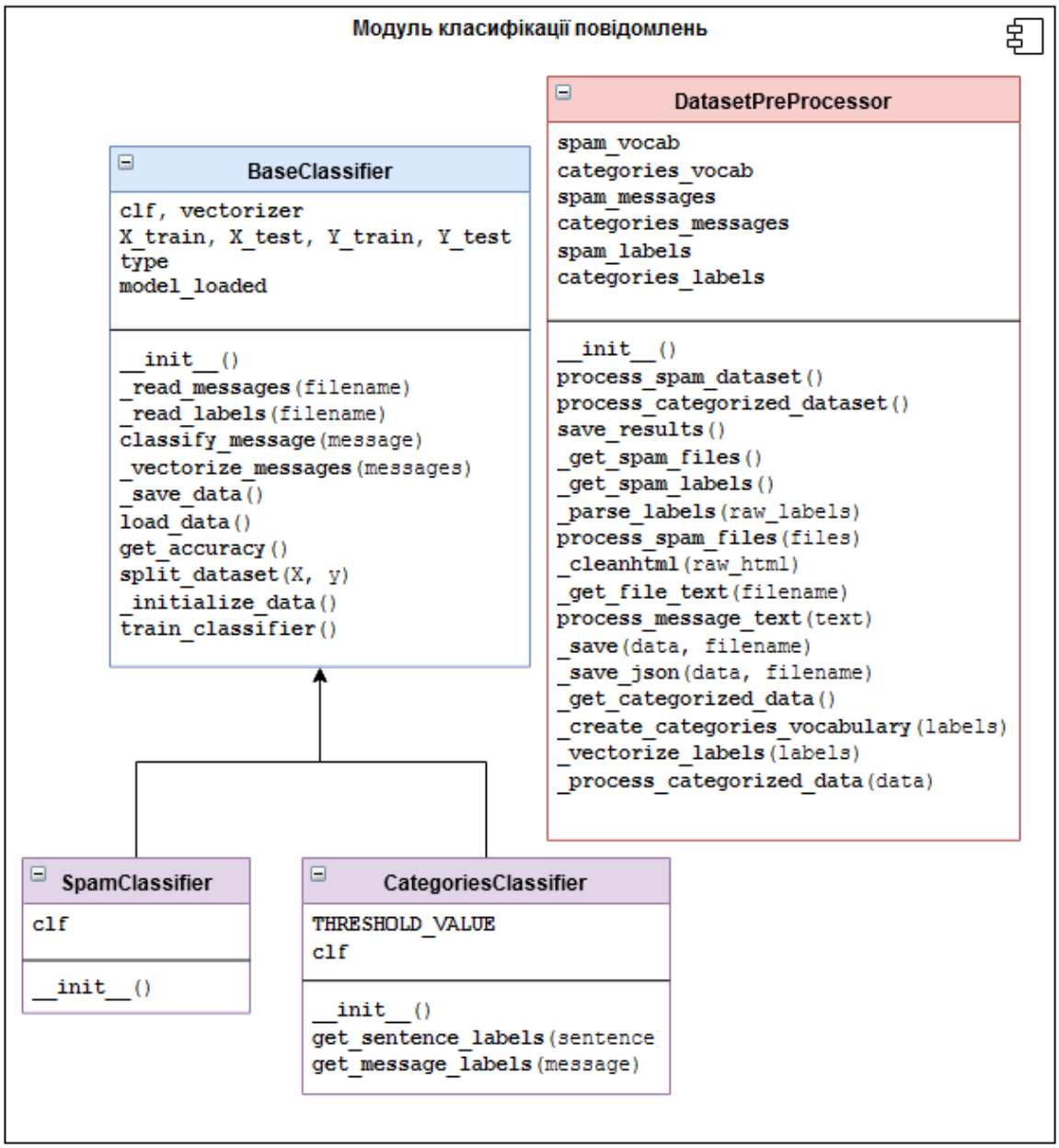


Рис. 3.6. Діаграма класів модуля класифікації повідомлень

Основними в спам класифікаторі є метод classify_message(), який приймає на вхід необроблене текстове повідомлення. Цей метод успадковується від базового класу та повертає 0 або 1, в залежності від того, класифіковане повідомлення як спам, чи ні.

В класифікаторі за категоріями основним є метод get_message_labels(). Завдяки атрибуту граничної ймовірності THRESHOLD_VALUE цей метод повертає список категорій до яких відноситься повідомлення.

Атрибут clf містить необхідний класифікатор. В залежності від класу, в clf може бути переданий класифікатор, що базується на методі опорних векторів або класифікатор Баєса.

Окрім модуля класифікації пакет допоміжних класів містить також наступні сутності:

1. ApplicationBlueprint – базовий клас для створення модулів та їх поєднання з головною аплікацією.
2. BaseView – базовий клас, від якого успадковуються представлення усіх модулів.
3. Login_required – декоратор для обмеження доступу до системи не аутентифікованим користувачам.
4. ControllerRegistry – клас, який забезпечує зв’язок контролерів з відповідними модулями.
5. ModelRegistry – клас, який забезпечує зв’язок моделей з відповідними модулями.
6. Email_builder – модуль для створення та відправлення повідомлення з підтвердженням реєстрації.

Основним модулем серверної частини є Mail_app. Цей модуль приймає запити, що стосуються повідомлень та їх типів. На рис. 3.7 зображена діаграма класів модуля Mail_app.

Шар представлень модуля Mail_app містить класи MessageView (відповідає за передачу повідомлень) та TypesView (відповідає за передачу типів). Для перевірки формату повідомлень, що приходять в запитах з клієнтської частини та конвертації даних реалізовано клас MessageSchema, атрибути якого відповідають полям повідомлення.

Шар контролерів містить лише один клас MessageController, який відповідає за створення, класифікацію та передачу повідомлень, а також їх типів до представлень.

При надходженні нового повідомлення класифікатор звертається до екземпляру класу спам-класифікатора, для його класифікації. У випадку,

якщо повідомлення помічається як спам, етап класифікації за категоріями пропускається. Якщо ж повідомлення пройшло фільтрацію, контролер звертається до екземпляру класу класифікатора за категоріями, що повертає список категорій. При поверненні порожнього списку, повідомлення помічається категорією “Unsorted”.

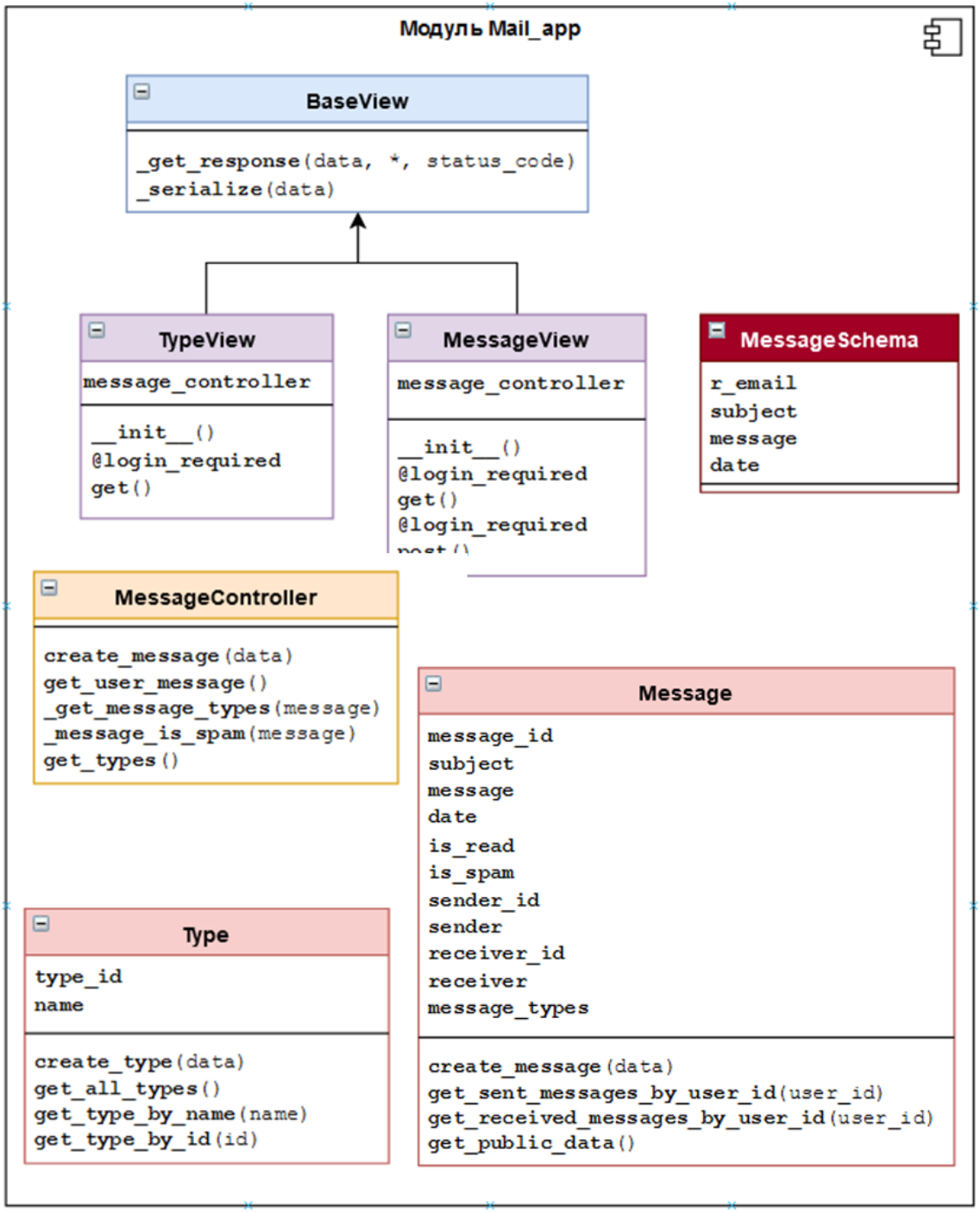


Рис. 3.7. Діаграма класів модуля Mail_app

Після класифікації повідомлення контролер звертається до відповідної моделі для додавання повідомлення в базу. Шар моделей містить класи Message та Type, які успадковуються від базової моделі db.Model з бібліотеки SQLAlchemy.

Іншим важливим модулем серверної частини є User_app. Він відповідає за виконання відповідних операцій з користувачем, таких як створення, оновлення, видалення та отримання інформації. На рис. 3.8 зображена діаграма класів модуля User_app.

Шар представлень модуля User_app складається з наступних класів:

1. UserView – відповідає за створення користувача, передачу його даних на клієнтську частину, оновлення даних профілю та зміну паролю.
2. UserAvatarView – відповідає за оновлення зображення користувача.
3. LoginView – відповідає за аутентифікацію користувача в системі.
4. LogoutView – відповідає за вихід користувача з системи.

Для валідації даних користувача, отриманих сервером використовуються класи UserRegisterSchema, UpdateUserSchema, LoginSchema та SocialLoginSchema.

В шарі контроллерів модуля User_app містяться класи UserController, який відповідає за обробку даних користувача, та LoginController, який відповідає за аутентифікацію та вихід з системи.

Користувач може бути аутентифікований за допомогою електронної пошти і паролю, а також за допомогою соціальної мережі Facebook або Google.

При аутентифікації користувача генерується ідентифікатор сесії, який зберігається в сховищі Redis для швидкого доступу, а також надсилається на клієнтську частину.

Якщо користувач аутентифікується за допомогою соціальних мереж, на сервер надходить запит з токеном та ім’ям провайдера (соціальної

мережі). Валідність токену перевіряється в LoginController за допомогою бібліотек, які надають інтерфейс до відповідної соціальної мережі. Якщо токен не валідний, сервер повертає помилку авторизації на клієнтську частину, в іншому випадку контролер отримує всю інформацію про користувача з соціальної мережі: ім'я, прізвище, зображення профілю, електронну пошту. Якщо користувач з отриманою електронною поштою не зареєстрований в системі, виконується додатковий крок: користувач додається до бази та активується. Після цього генерується ідентифікатор сессії та надсилається на клієнтську частину.

Шар моделей містить лише один клас – User, який є об'єктно-реляційним відображенням таблиці User_profile в базі даних.

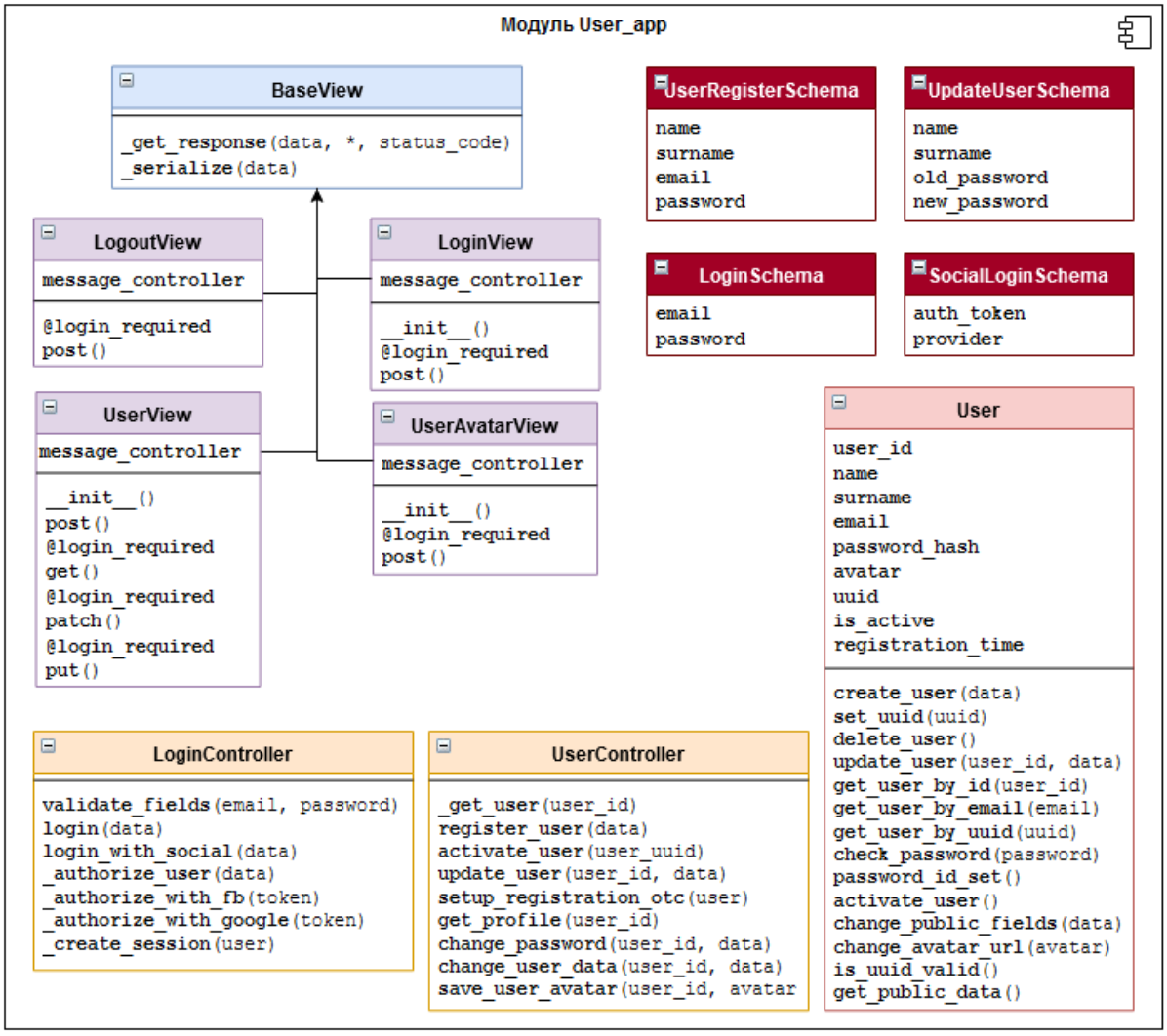


Рис. 3.8. Діаграма класів модуля User_app

Модуль OTC_app відповідальний за генерацію тимчасового коду для підтвердження реєстрації користувача. Він містить лише один клас контролера OtcController, базовий клас тимчасового коду BaseOtc, а також його спадкоємця RegistrationOtc.

OTC_app не відправляє запитів та не має відображення у базі даних, тому в цьому модулі відсутні шари представлення та моделей. При реєстрації користувача OtcController записує в сховище Redis унікальний ключ, необхідний для активації користувача. Після активації ключ видаляється а статус користувача в базі даних змінюється на “активний”. На рис. 3.9 зображена діаграма класів модуля OTC_app.

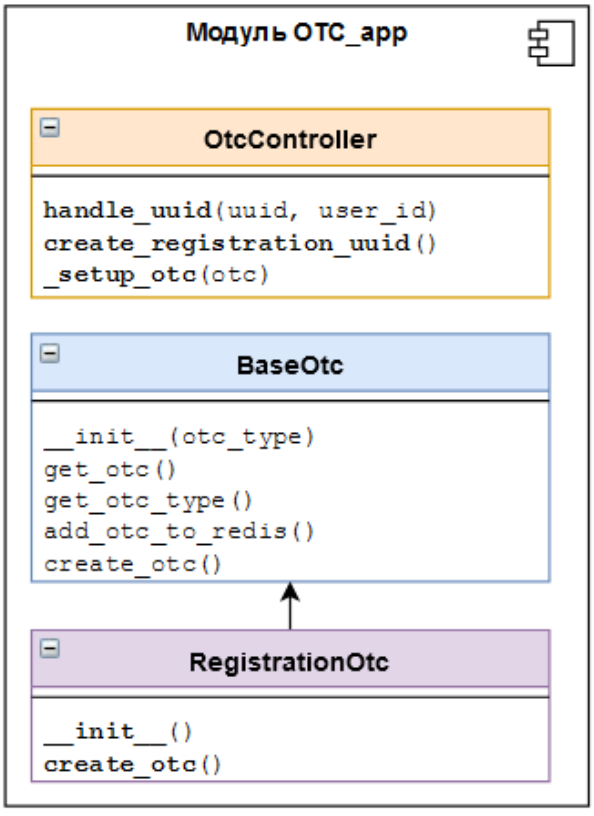


Рис. 3.9. Діаграма класів модуля OTC_app

З моделей веб-додатку бібліотекою SQLAlchemy автоматично сформовано структуру бази даних, після чого вона була перенесена в базу mail_db в СКБД PostgreSQL. На рис. 3.10 зображена структура створеної бази даних.

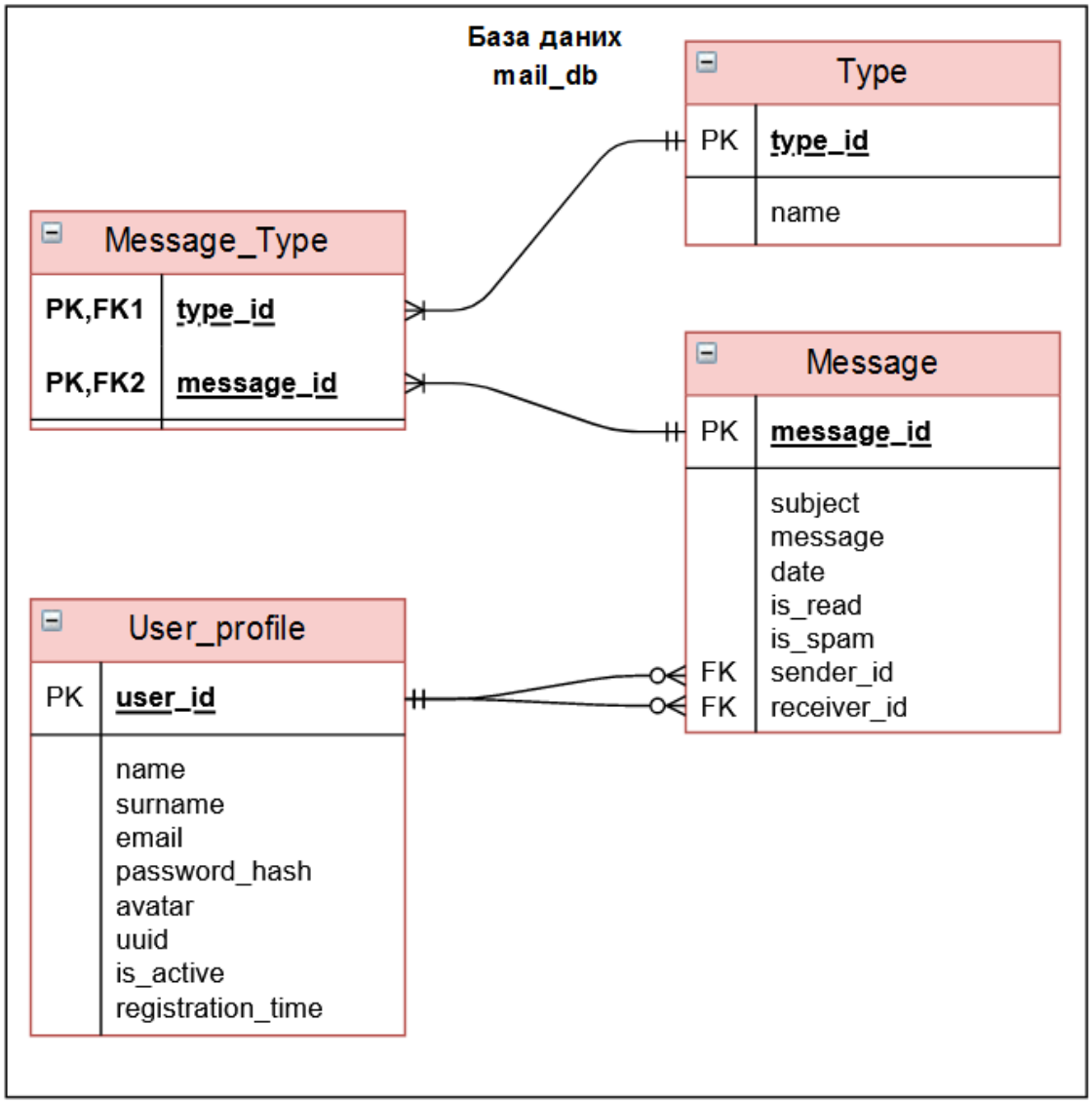


Рис. 3.10. Структура бази даних mail_db

*3.2.2 **Клієнтська частина***

Клієнтська частина розроблена з використанням фреймворку Angular. Додаток складається з компонентів, сервісів та додаткових модулів які пов’язані між собою. Усі компоненти ініціалізуються в батьківському компоненті клієнтської частини

Усі компоненти веб-додатку містять ts файл, в якому описується логіка обробки даних компоненту, html файл, що описує інтерфейс користувача для взаємодії з компонентом, css файл, який описує стилі для зовнішнього відображення, а також spec.ts файл для створення тестів відповідного компоненту.

В клієнтській частині створено наступні компоненти:

1. App component.
2. Main-page component.
3. Message-list component.
4. Register-pop-up component.
5. User-profile component.
6. Email-confirmation component.
7. Header component.
8. Footer component.
9. Error component.

Майже усі компоненти мають взаємодіяти один з одним, а також відправляти необхідні дані на сервер. Логіка зв’язку компонентів між собою та з сервером реалізована у сервісах. Окрім цього, сервіси виконують обробку даних перед їх використанням компонентами або відправленням запиту до сервера. При розробці реалізовано наступні сервіси:

1. Mail service.
2. User service.
3. Auth service.
4. Error service.

Окрім компонентів та сервісів також реалізовано модулі App-routing module та Auth guard.

App-routing module забезпечує можливість доступу користувача до компонентів системи за посиланнями.

Модуль Auth guard обмежує доступ не аутентифікованого користувача до необхідних посилань в App-routing module, перенаправляючи його на головну сторінку з вікном аутентифікації. На рис. 3.11 зображена структура клієнтської частини веб-додатку.

Основним компонентом є Message-list component. Він описує сторінку користувача з повідомленнями. Для отримання даних про повідомлення користувача, Message-list component звертається до Mail service, який робить

асинхронний GET запит до сервера і повертає список повідомлень користувача.

Отримані повідомлення розділені на основні типи:

1. Received – отримані повідомлення.
2. Sent – надіслані повідомлення.
3. Spam – спам повідомлення.

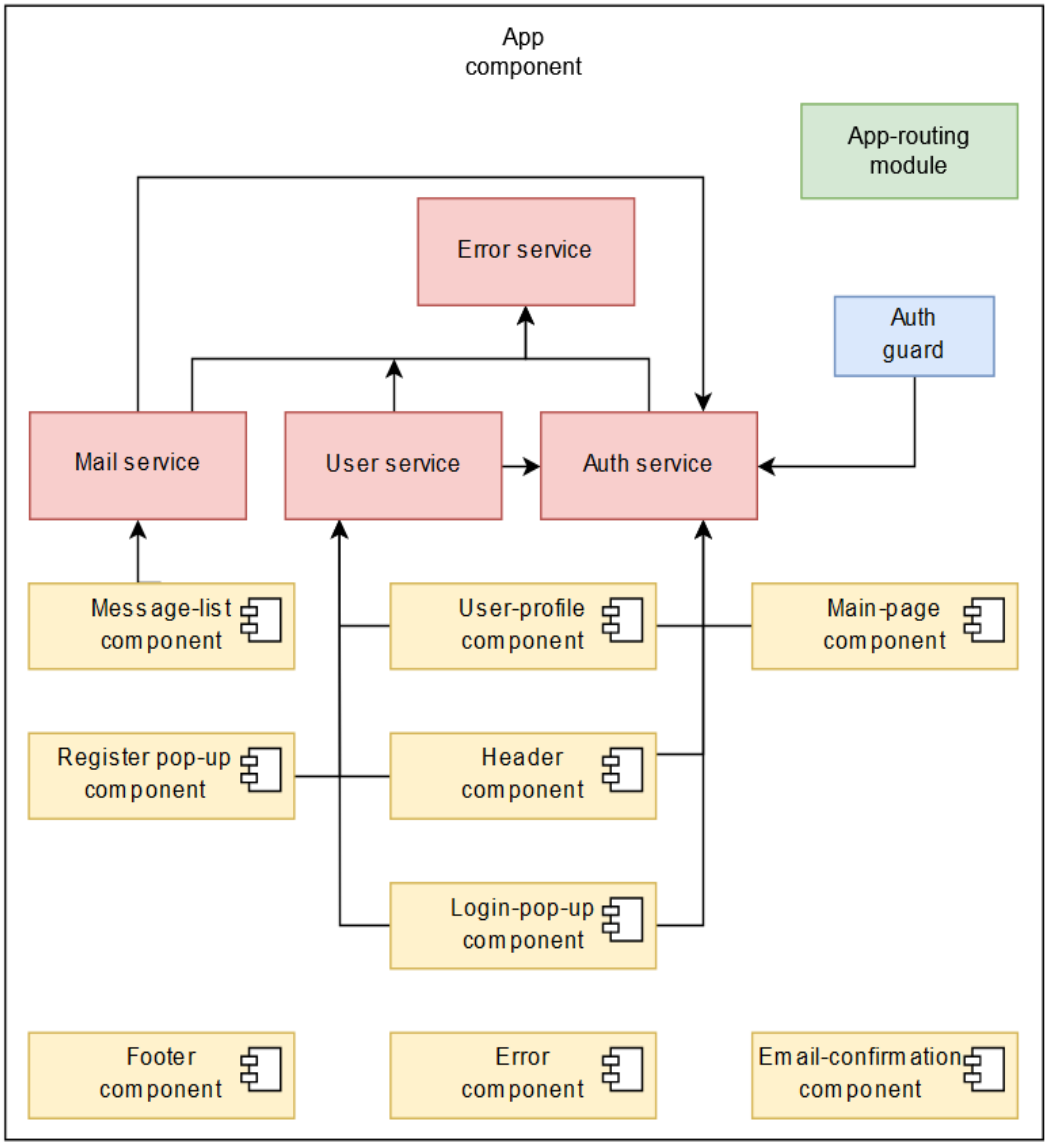


Рис. 3.11. Структура клієнтської частини веб-додатку

Тип “Received”, в свою чергу, поділяється на категорії повідомлень, визначені класифікатором, за якими вони розділяються при відображенні.

Окрім отримання повідомлень користувача, Message-list component дозволяє надіслати нове повідомлення. При виклику методу createMessage(), компонент надсилає дані з відповідних полів до Mail service, який надсилає асинхронний POST запит до сервера.

Іншим важливим компонентом є User-profile component. Він відповідає за відображення та редагування даних користувача. Цей компонент звертається до сервісу User service, який надає можливість надсилати запити на отримання та оновлення даних про користувача, таких як пароль, зображення, ім’я та прізвище.

Компоненти Register-pop-up, та Email-confirmation відповідають за створення, підтвердження користувача в системі. Як і User-profile component вони звертаються до сервісу User service для відправлення відповідних запитів.

Компонент Login-pop-up надає можливість користувачу аутентифікуватися в веб-додатку. Існує два типи аутентифікації:

1. За допомогою електронної пошти та паролю.
2. Через соціальні мережі Facebook та Google.

При звичайній аутентифікації на сервер надсилається email та пароль користувача. При аутентифікації через одну з соціальних мереж на клієнтську частину приходить відповідний токен, який надсилається на сервер для повторної валідації.

Компоненти Main-page, Header, Footer та Error є допоміжними, їх структура аналогічна до вже згаданих компонентів.

Auth service відповідає за аутентифікацію користувача. Якщо User service надсилає запит на аутентифікацію користувача, при коректних даних сервер повертає відповідь з ідентифікатором сесії, який записується в локальне сховище даних браузера. Поки цей ідентифікатор присутній в сховищі, а також співпадає з відповідним ідентифікатором на сервері, користувач може отримувати доступ до будь-якого компоненту клієнтської частини.

При виході користувача з системи, ідентифікатор видаляється зі сховища, а також на сервері. Максимальний час існування ідентифікатора – 24 години, після чого він також видаляється.

Кожен з сервісів звертається до Auth service для підтвердження того, що користувач аутентифікований. У випадку, якщо ідентифікатор в клієнтському сховищі виявився не валідним, він видаляється, закриваючи доступ користувачу до компонентів системи.

3.3. **Опис функціональних можливостей системи**

До основних вимог розробленого веб-додатку відносяться: реєстрація користувача, аутентифікація користувача в системі, редагування профілю користувача, класифікація повідомлень користувача, відправлення користувачем повідомлень.

Не аутентифікований користувач має доступ лише до головної сторінки веб-додатку. На головній сторінці користувач може викликати вікно реєстрації або авторизації, натиснувши відповідну кнопку. При спробі неавторизованого користувача отримати доступ до системи по прямому посиланню, його буде перенаправлено на головну сторінку.

Користувач може зареєструватися в системи, натиснувши кнопку “Sign up” на головній сторінці системи, в спливаючому вікні заповнивши необхідну для реєстрації інформацію та натиснувши кнопку “Register”. Після цього на вказану електронну пошту користувача прийде посилання на підтвердження реєстрації. Проходження за цим посиланням активує профіль користувача в системі.

Для аутентифікації користувач має натиснути кнопку “Sign in” на головній сторінці системи, після чого ввести свою електронну пошту та пароль і натиснути “Login”. У випадку невірного пароля або якщо користувач з такою електронною поштою не існує в системі йому не буде надано доступ до системи.

Користувач також може пройти аутентифікацію через соціальні мережі Facebook або Google. Для цього необхідно у вікні аутентифікації натиснути одну з кнопок: “Facebook” або “Google”. Після цього необхідно пройти аутентифікацію в обраній соціальній мережі. У випадку, якщо не зареєстрований у системі користувач аутентифікується за допомогою однієї з соціальних мереж, його профіль автоматично буде створено, активовано і заповнено інформацією з обраної соціальної мережі.

Аутентифікований користувач може переглядати свої повідомлення. Для цього необхідно з головної сторінки перейти на сторінку повідомлень, натиснувши кнопку “My mail”, або скориставшись прямим посиланням “/messages”. На сторінці повідомлень користувач може переглядати повідомлення з трьох розділів: “Received”, “Sent”, “Spam”. Для цього необхідно натиснути на випадаюче меню “Section” та обрати потрібний розділ. В розділі “Received” користувач може переглядати повідомлення з різних категорій, натиснувши на необхідну категорію. Повідомлення, які не потрапили до жодної з основних категорій, потрапляють в категорію “Unsorted”.

Аутентифікований користувач може відправляти повідомлення іншим користувачам. Для цього потрібно на сторінці повідомлень натиснути кнопку “+”, в випадаючій панелі заповнити необхідні поля повідомлення та натиснути кнопку “Send”.

Аутентифікований користувач може редагувати свій профіль. Для цього необхідно натиснути на значок профілю, після чого натиснути кнопку “Edit” в боковій панелі, змінити необхідні поля та натиснути кнопку “Submit Changes”. Для зміни паролю користувачу необхідно в боковій панелі натиснути кнопку “Change password” та заповнити необхідні поля. У випадку, якщо користувач аутентифікувався в системі за допомогою соціальних мереж і не змінював пароль, поле підтвердження старого паролю відсутнє. На рис. 3.12 зображена Use Case діаграма для розробленого додатку.

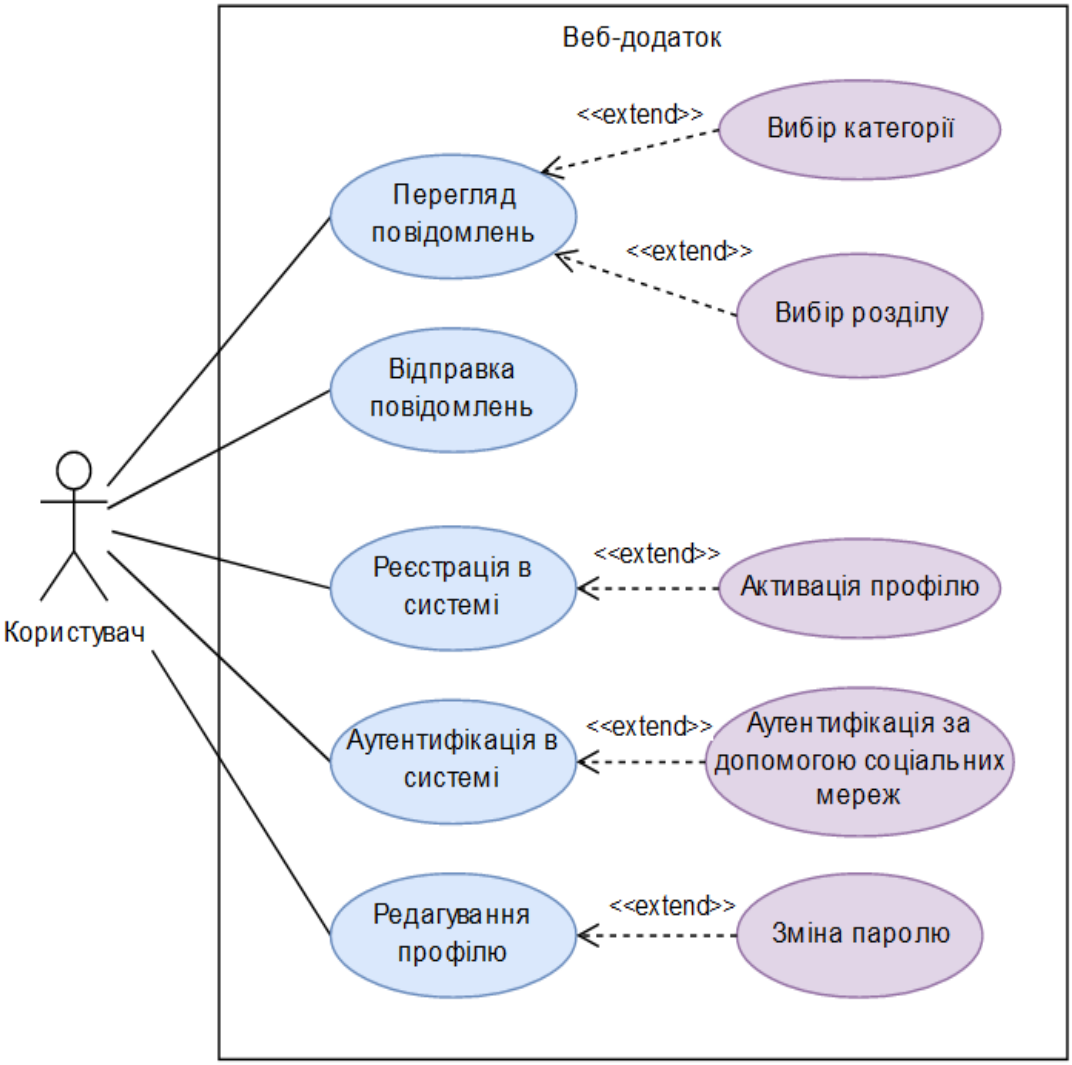


Рис. 3.12. Use Case діаграма веб-додатку

3.4. **Висновки до розділу 3**

В даному розділі обрано технології для розробки веб-додатку, який реалізовує комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Визначено загальну архітектуру системи, розглянуто її основні компоненти. Описано структуру серверної частини, її основні та допоміжні модулі. Розглянуто модуль класифікації повідомлень, в якому реалізовані метод опорних векторів для фільтрації спаму та метод Баєса для класифікації повідомлень за категоріями. Детально розглянуті усі представлення, контролери на моделі серверної частини. Сформовано діаграми класів основних модулів серверної частини. Схематично зображено структуру бази

даних. Описано основні компоненти та сервіси клієнтської частини, зв’язки між ними, а також їх взаємодію з сервером.

Також розглянуто функціональні можливості веб-додатку. Детально описано, а також представлено у вигляді Use Case діаграми взаємодію користувача з системою.

Розроблений веб-додаток надає користувачу можливість обмінюватися електронними повідомленнями в межах системи. Класифікація текстового вмісту повідомлень проходить на сервері непомітно для користувача при відправленні. Результат класифікації відображається на сторінці повідомлень користувача. Повідомлення, помічені як спам, не класифікуються і відображаються у вигляді окремого розділу.

4. ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Процес тестування програмної реалізації комбінованого методу автоматизованої класифікації розділено на три етапи:

1. Тестування спам-класифікатора.
2. Тестування класифікатора за категоріями.
3. Тестування комбінованої класифікації.

На кожному етапі виміряно точність та швидкість класифікації, а також швидкість тренування при різних параметрах класифікаторів, а також різних розмірах навчальних вибірок.

Точність класифікації визначається відношенням правильно класифікованих повідомлень з тестової частини вибірки до загальної кількості повідомлень тестової частини.

Для порівняння точності класифікації обрано три додаткових методи машинного навчання:

1. Найближчий центроїд.
2. Дерево рішень.
3. Випадковий ліс.

При класифікації текстових даних на точність можуть впливати такі параметри, як розмір навчальної та тестової вибірки, а також випадковий стан.

Значення випадкового стану впливає на те, як дані вибірки будуть перемішані при розділенні на навчальну та тестову вибірку. При незмінних вибірках однакові значення випадкового стану повертають однаково перемішані вибірки у випадку повторного запуску.

У випадку класифікації за категоріями до змінних параметрів додано також граничне значення ймовірності.

Також порівняно швидкість класифікації при використанні серіалізації і десеріалізації моделей з повторним тренуванням для перевірки необхідності зберігання моделей.

4.1. **Тестування спам класифікатора**

Для першого тесту обрано розбиття вибірки повідомлень наступним чином: 825 навчальних повідомлень, 1675 тестових, а також значення випадкового стану – 42. Результати тесту наведені у табл. 4.1.

Таблиця 4.1

Результати першого тесту класифікатора спаму

	Розроблений класифікатор спаму	Дерево рішень	Випадковий ліс	Найближчий центроїд
Точність класифікації	0.8853	0.8310	0.8502	0.8460
Час навчання	14.3559 с.	1.2337 с.	2.4764 с.	0.6008 с.
Час класифікації без серіалізації	14.3791 с.	1.2406 с.	2.4890 с.	0.6079 с.
Час класифікації З серіалізацією	0.6126 с.	05049 с.	0.5272 с.	0.4598 с.

Хоча розроблений спам-класифікатор, який базується на методі опорних векторів має найгірший час навчання та класифікації, його точність класифікації найвища.

За результатами першого тесту також видно, що при серіалізації моделі класифікатора час класифікації лінійний. Це є особливо критичним у розробленому класифікаторі з часом класифікації без серіалізації 14.3791 секунд. Поганий час навчання Розроблений класифікатор показує через велику кількість ітерацій. Класифікатори дерева рішень та випадкового ліс мають значно меншу кількість кроків завдяки невеликій кількості класів до яких необхідно віднести повідомлення. Класифікатор найближчого

центроїда використовує єдину формулу для класифікації, що надає можливість робити обчислення за лінійний час.

Для другого тесту тесту змінено розбиття вибірки повідомлень. Кількість навчальних і тестових повідомлень обрана рівною – 1250 повідомлень в кожній вибірці. Оскільки вибірка змінилась, нема необхідності змінювати значення випадкового стану. Результати другого тесту наведені у табл. 4.2.

Таблиця 4.2

Результати другого тесту класифікатора спаму

	Розроблений класифікатор спаму	Дерево рішень	Випадковий ліс	Найближчий центроїд
Точність класифікації	0.9095	0.8514	0.8764	0.8240
Час навчання	28.0267 с.	1.4617 с.	3.507 с.	0.6269 с.
Час класифікації без серіалізації	28.0541 с.	1.5200 с.	3.520 с.	0.6337 с.
Час класифікації З серіалізацією	0.6565 с.	0.5115 с.	0.5235 с.	0.5032 с.

Результати другого тесту показали ріст точності усіх класифікаторів, окрім найближчого центроїда. У випадку розробленого класифікатора при збільшенні навчальної частини вибірки в 1.5 рази, час його навчання зріс вдвічі.

Час класифікації з використанням серіалізації моделі класифікатора майже не змінився, що підтверджує необхідність зберігання моделей.

Для третього тесту спам класифікатора обрано розмір навчальної частини вибірки – 1675 повідомлень, а тестової – 825 повідомлень. Результати останнього тесту наведені у табл. 4.3.

В третьому тесті класифікатор найближчого центроїда показує найгірший результат за точністю. Інші класифікатори покращили точність класифікації.

Найкращу точність класифікації – 0.9 має розроблений класифікатор спаму, який базується на методі опорних векторів. Хоча цей метод має найгірший час навчання, використання серіалізації для збереження моделі позбавляє необхідності проводити навчання кожного разу при перезавантаженні сервера.

Таблиця 4.3

Результати третього тесту класифікатора спаму

	Розроблений класифікатор спаму	Дерево рішень	Випадковий ліс	Найближчий центроїд
Точність класифікації	0.9206	0.8742	0.8802	0.8060
Час навчання	44.8413 с.	2.7435 с.	4.559 с.	0.6587 с.
Час класифікації без серіалізації	44.8742 с.	2.7499 с.	4.5726 с.	0.6655 с.
Час класифікації З серіалізацією	0.6774 с.	0.5283 с.	0.5244 с.	0.5042

За результатами трьох тестів розроблений класифікатор спаму, який базується на методі опорних векторів, показав найкращу точність класифікації. Окрім цього, виявлено, що найбільш оптимальним є розділення вибірки на навчальну та тестову за відношенням 0.67/0.33.

При використанні серіалізації та десеріалізації моделей, швидкість навчання класифікатора та збільшення розміру навчальної частини вибірки майже не впливає на час класифікації. Завдяки цьому розроблений класифікатор спаму може працювати з такою ж швидкістю, як наведені в тестах класифікатори, але мати при цьому більшу точність.

4.2. **Тестування класифікатора за категоріями**

Для перевірки класифікатора за категоріями вирішено обрати оптимальне відношення для розділення вибірки, визначене за минулим тестом. Загальна кількість повідомлень у вибірці – 7600. Розмір навчальної частини вибірки – 5100 повідомлень, розмір тестової частини – 2500 повідомлень.

В першому тесті вирішено порівняти швидкість навчання та класифікації. Оскільки граничне значення ймовірності не впливає на час роботи класифікаторів, порівняння точності класифікації винесено в другий тест. Результати порівняння швидкості роботи класифікаторів наведено у табл. 4.4.

Таблиця 4.4

Результати тесту швидкості класифікатора за категоріями

	Розроблений Класифікатор за категоріями	Дерево рішень	Випадковий ліс	Найближчий центроїд
Час навчання	1.4064 с.	11.7741 с.	18.4289 с.	0.6411 с.
Час класифікації без серіалізації	1.4136 с.	11.7799 с.	18.4411 с.	0.6476 с.
Час класифікації з серіалізацією	0.4483 с.	0.4555 с.	0.4785 с.	0.4793

За результатами першого тесту найповільнішими при класифікації без серіалізації моделей виявились класифікатори дерева рішень та випадкового лісу. При використанні серіалізації усі класифікатори показують майже однакові результати в межах від 0.44 до 0.48 секунд.

Для другого тесту вирішено обрати три значення граничної ймовірності: 0.7, 0.75 та 0.8. Оскільки ці значення впливають лише на точність розробленого класифікатора за категоріями, результати інших класифікаторів будуть незмінними. Результати другого тесту класифікатора за категоріями наведено у табл. 4.5.

Таблиця 4.5

Результати тесту точності класифікатора за категоріями

Граничне значення ймовірності	Розроблений класифікатор за категоріями	Дерево рішень	Випадковий ліс	Найближчий центроїд
0.7	0.8797	0.7393	0.8505	0.8011
0.75	0.8976	0.7393	0.8505	0.8011
0.8	0.8913	0.7393	0.8505	0.8011

В другому тесті розроблений класифікатор за категоріями показав найкращу точність класифікації при усіх граничних значеннях ймовірності. Найбільшої точності класифікації – 0.8976 досягнуто при граничному значення ймовірності 0.75.

4.3. **Тестування комбінованої класифікації**

Оскільки в комбінованому методі використовується два класифікатори, значення точності комбінованого методу буде обчислене як добуток значень точності обох класифікаторів.

Для знаходження точності класифікаторів, обраних для порівняння, навчальні вибірки і результуючі класи будуть об’єднані в одну вибірку, на якій буде проводитись тренування і перевірка точності.

Для тесту комбінованої класифікації також порівняні такі характеристики, як час навчання та класифікації. Для досягнення мінімального часу класифікації використано серіалізацію та десеріалізацію навчених моделей. Результати тесту комбінованої класифікації наведено у табл. 4.6.

Таблиця 4.6

Результати тесту комбінованого методу

	Комбінований метод	Дерево рішень	Випадковий ліс	Найближчий центроїд
Точність	0.8263	0.7367	0.7512	0.648
Час навчання	46.2477 с.	16.2830 с.	35.8102 с.	0.7123 с.
Час класифікації	0.4584 с.	0.4825 с.	0.5197 с.	0.4613 с.

За результатами тесту комбінованої класифікації визначено, що розроблений комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти має набагато кращу точність класифікації, в порівнянні з іншими методами, в яких не розділені етапи фільтрації та класифікації за категоріями.

Хоча в комбінований методі досить повільний час тренування, можливість зберігати моделі позбавляє необхідності виконувати тренування моделі при кожному запуску сервера. На рис. 4.1 зображений графік залежності часу підготовки моделей від розміру вибірки. З ростом кількості повідомлень у вибірці час завантаження і десеріалізації моделі майже не змінюється.

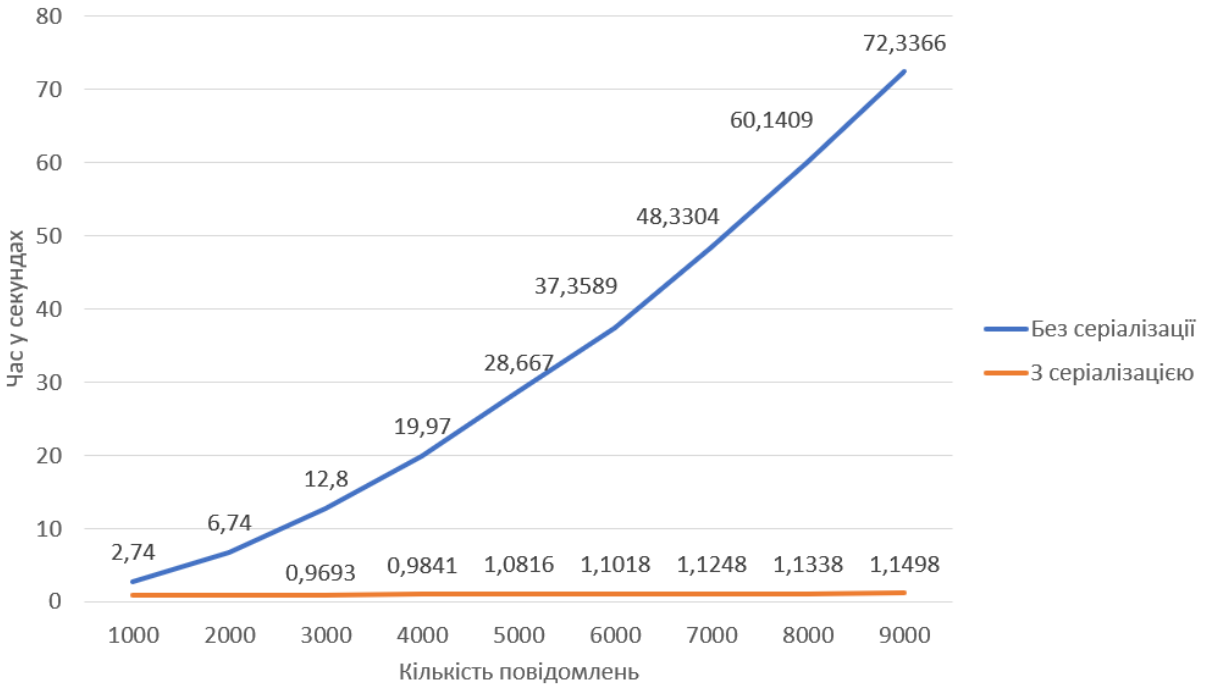


Рис. 4.1 Графік залежності часу підготовки моделей від розміру вибірки

4.4. **Висновки до розділу 4**

В даному розділі проведено аналіз ефективності програмної реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Порівняно час, а також точність класифікації повідомлень з іншими методами машинного навчання. Проведене тестування поділено на три етапи: тестування класифікатора спаму, тестування класифікатора за категоріями та тестування реалізованої комбінованої класифікації. Основними параметрами, які змінювались під час тестування були: розмір навчальної та тестової частини вибірки та граничне значення ймовірності.

Також перевірено ефективність серіалізації та десеріалізації моделей, для уникнення повторного тренування при ініціалізації моделей.

Визначено, що реалізований комбінований метод надає можливість виконувати більш точну класифікацію текстового вмісту електронних повідомлень, а також має досить високу швидкість класифікації, завдяки чому може бути використаний для класифікації повідомлень в реальному часі.

5. ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

5.1. Опис проблеми

Однією з найважливіших галузей в сучасному світі є автоматизована обробка природної мови, яка є загальним напрямом штучного інтелекту та математичної лінгвістики. Обробка природної мови допомагає вирішити такі проблеми, як неспроможність комп'ютера аналізувати мову, а також формувати наділені сенсом речення [8].

Однією з підзадач обробки природної мови є класифікація тексту, яка полягає в віднесені текстових даних до певного класу за його ознаками. Ознаками тексту можуть бути його мова, цілі речення та окремі слова.

З появою Інтернету, виникла можливість обмінюватися текстовими повідомленнями, використовуючи електронну пошту. На сьогоднішній день цей спосіб передачі повідомлень є одним з найпоширеніших. Електронною поштою користуються для ведення бізнес-листування, розповсюдження реклами, надсилання сповіщень, спілкування тощо. Але виникнення та розвиток електронних поштових систем не вирішило усіх проблем, адже з плином часу через значне збільшення кількості повідомлень, які користувач кожного дня, а також появу спам-розсилок, з'явилась потреба у класифікації повідомлень. Звичайні системи зазвичай надають лише можливість фільтрації повідомлень від спаму, але на сьогоднішній день цього не достатньо для ведення чистої поштової скриньки, в якій користувачу буде легко орієнтуватись.

Нажаль, на сьогоднішній день класифікатори, які надають аналізуючи текст повідомлення надають можливість відносити його до різних категорій використовуються лише на різних змаганнях, що проходять на таких платформах, як Kaggle, а також у закритих або вузькоспеціалізованих системах.

В сучасних поштових системах за допомогою класифікаторів фільтруються спам-повідомлення. Деякі з них також виконують сортування повідомлень за невеликою кількістю категорії, використовуючи метадані,

але цей спосіб не можна назвати універсальним, оскільки не всі повідомлення електронної пошти можуть бути класифіковані за їх метаданими. Через це користувачу часто доводиться шукати необхідне повідомлення серед десятків зайвих рекламних повідомлень та сповіщень від соціальних мереж.

Іншим недоліком є те, що у випадку присутності в електронної поштовій системі класифікації повідомлення по метаданим, користувач має точно знати до якої саме категорії це повідомлення буде віднесено, оскільки воно потрапить лише в одну категорію, не маючи можливості потрапити в будь-яку з інших.

Крім того, в деякі електронні поштові системи повідомлення сортуються за категоріями, непотрібними користувачу. У випадку потрапляння важливого повідомлення до однієї з таких категорій, користувач може навіть не помітити його, оскільки він не зацікавлений в перегляді цієї категорії взагалі.

Хоча пошук найзручнішої електронної поштової системи є вирішенням частини проблем для конкретного користувача, це рішення не є універсальним. Такий пошук може займати багато часу та не завжди бути успішним. Більш практичним для вирішення вказаної проблеми є надання можливості користувачу самому обирати категорії, які є для нього необхідними.

В деяких системах електронної пошти розділення повідомлень за категоріями відсутнє зовсім. Хоча таке рішення позбавляє недоліку помилкової класифікації, пошук необхідного повідомлення може займати дуже багато часу через їх невпорядкованість.

Частково проблему загублених повідомлень вирішує можливість текстового пошуку але таке рішення актуальне лише за умови, що користувач заздалегідь знає за якими ключовими словами необхідно виконувати пошук.

Усі існуючі в сучасних системах електронної пошти проблеми вкупі з швидким ростом кількості користувачів, а також об’ємів даних що передаються, погіршують досвід використання електронної пошти користувачами. Це, в свою чергу вказує на необхідність вирішення існуючих проблем. На рис. 5.1 зображена схема дерева проблем.

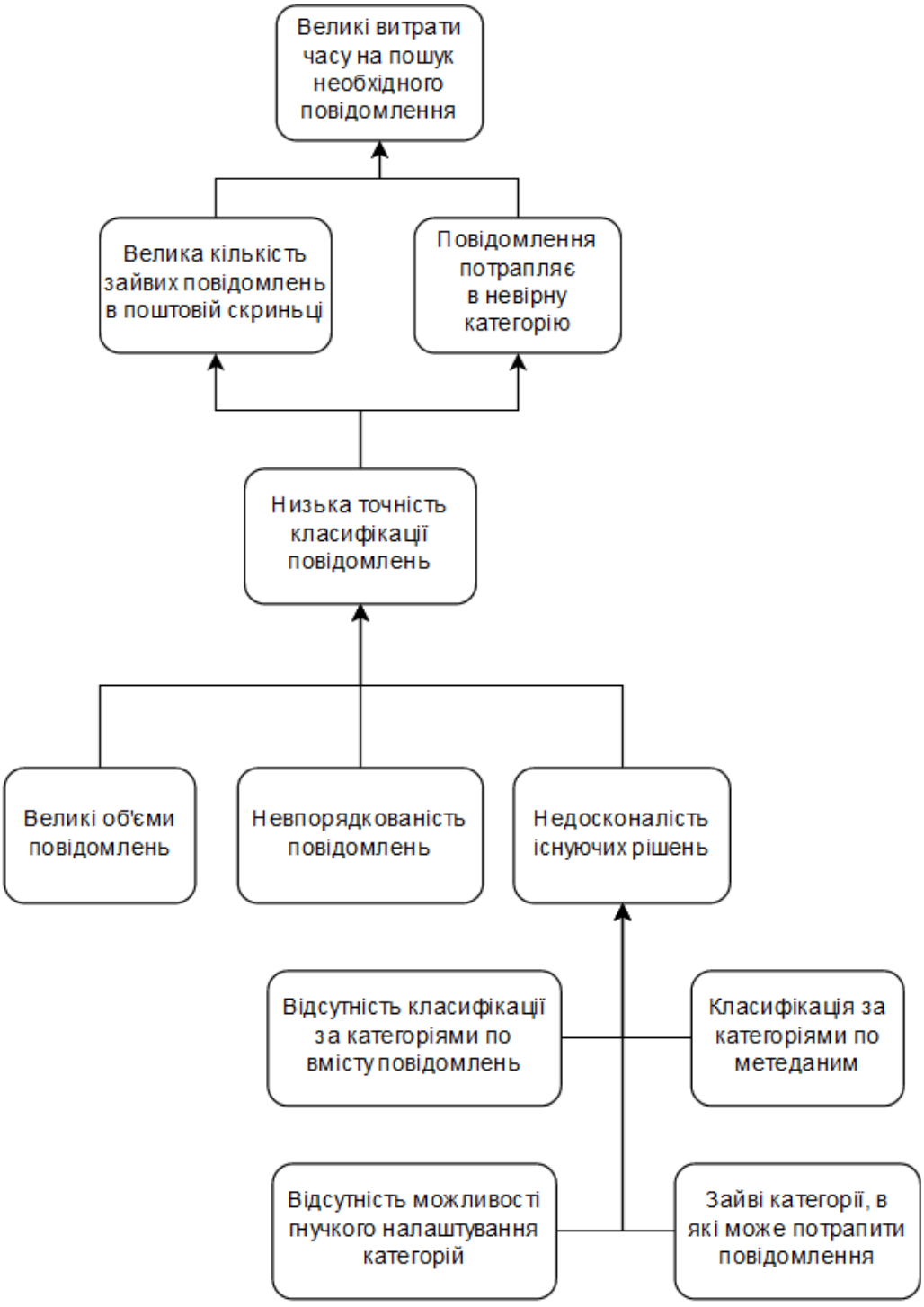


Рис. 5.1. Схема дерева проблем

5.2. **Зацікавлені сторони**

У вирішенні проблеми недостатньої точності автоматизованої класифікації текстового вмісту повідомлень електронної пошти є декілька зацікавлених сторін.

Найголовнішою з зацікавлених сторін є власники поштових систем, через які здійснюється обмін електронними повідомленнями, а саме Google, Yahoo, Mail.ru, Microsoft, Yandex та інші. Конкуруючи між собою, всі ці компанії зацікавлені у якомога точнішій класифікації повідомлень, та наданні найбільш зручного інтерфейсу, оскільки ці фактори є найважливішими для користувачів. Вплив цієї сторони є високим, адже саме власники поштових систем мають безпосередній вплив на них, займаються їх підтримкою та інтеграцією нових функцій.

Звичайно, компанія Google зараз поза конкуренції, адже в її поштовій системі зареєстрований найбільший відсоток користувачів усього світу, але ситуація може змінитися, якщо одна з поштових систем, що входить до списку найпопулярніших інтегрує можливість класифікації повідомлень за їх вмістом на користувацькому рівні

Іншою зацікавленою стороною є власники різних інформаційних ресурсів, магазинів та соціальних мереж, адже для них важливо, щоб поштова система правильно класифікувала повідомлення з їх розсилок. Як приклад, правильна класифікація надасть можливість відносити повідомлення від сайту з новинами не просто до категорії “Новини” а визначати, до якого саме типу новин відноситься повідомлення: “Спорт”, “Бізнес” або “Наука”. У випадку, якщо користувач, перевіряючи повідомлення певної категорії, натрапить на повідомлення від інформаційного ресурсу, що не відноситься до неї, таке повідомлення скоріше за все буде пропущене. Аналогічно, якщо повідомлення не буде відображене в необхідній категорії, зацікавлений користувач може пропустити його. Усе це впливає на відвідуваність ресурсу, що є головним для його власника. Вплив власників інформаційних ресурсів досить

слабкий, адже вони не приносять прибуток, якщо не замовляють рекламу, а також не впливають на користувацьку базу поштової системи.

Ще одною важливою зацікавленою стороною є рекламодавці. Зазвичай, рекламні повідомлення виносяться на верхній рівень категорії “Реклама”, а також позначаються відповідним ярликом. В такому випадку користувач не зацікавлений в перегляді цих повідомлень і зазвичай одразу їх видаляє.

При правильній класифікації рекламних повідомлень ймовірність того, що користувач його перегляне підвищується. Так повідомлення про знижки на техніку може бути віднесено не до категорії “Реклама”, а до категорії “Техніка”. На сьогоднішній день класифікація таких повідомлень реалізується шляхом метаданих, не аналізуючи вміст повідомлення. Вплив рекламодавців є середнім, оскільки вони є основним джерелом прибутку власників поштових систем.

Четвертою, та найбільшою за кількістю зацікавленою стороною є користувачі електронних поштових систем. Саме завдяки популярності серед користувачів поштові системи існують і розвиваються, а також залучають нових рекламодавців. Звичайний користувач зацікавлений в декількох речах.

По-перше, класифікація повідомлень має бути дуже точною. Спам повідомлення не повинні потрапляти до основної категорії, а також повідомлення не мають бути віднесені до не релевантних категорій.

По-друге, в системі має бути легко знайти будь-яке повідомлення. Пошук повідомлення, необхідного користувачу не повинен займати багато часу, а тим більш необхідності в процесі перевіряти зайві категорії. Вплив користувача є середнім, оскільки їх кількість визначає популярність електронної поштової системи, що ключовим фактором для рекламодавців, а також сервісів, що аналізують дані. Список зацікавлених сторін, їх інтерес та вплив зазначені в табл. 5.1.

Таблиця 5.1

Основні зацікавлені сторони

Зацікавлена сторона	**Інтерес зацікавленої сторони**	**Вплив зацікавленої сторони**	**Стратегія приваблення зацікавлених сторін**
Компанії-власники поштових систем	Надання точної класифікації повідомлень та зручного інтерфейсу.	Високий	Проведення маркетингової компанії Розміщення на тематичних виставках. Виступ на конференціях з доповіддю. Проведення презентації для представників зацікавлених сторін
Власники інформаційних ресурсів	Поява повідомлень у відповідних категоріях	Низький	
Рекламодавці	Актуальність рекламних об’яв для користувачів електронних поштових систем	Середній	
Користувачі	Сортування повідомлень за категоріями, інтуїтивний інтерфейс, точність класифікації	Середній	

5.3. **Комерційне рішення. Основні характеристики**

Враховуючи всі вказані та описані існуючі проблеми, можна описати кінцевий продукт, метою якого є їх вирішення. Даний програмний продукт буде реалізовувати комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти. Цей метод надає

можливість підвищити точність класифікації, завдяки розділенню етапів фільтрації та класифікації за категоріями, а також модифікації методу Баєса, який застосовується на другому етапі. На відміну від існуючих методів, для класифікації використано комбінацію двух методів, а також введено граничне значення ймовірності для віднесення повідомлення до декількох категорій.

Особливо помітними зміни будуть для користувачів поштових систем, оскільки зменшиться кількість невірно класифікованих повідомлень. Завдяки класифікації за декількома категоріями, використання електронної пошти стане більш інтуїтивним та гнучким для користувачів.

Для рекламодавців результат буде менш помітним. Завдяки класифікації повідомлень за їх текстовим вмістом, відвідуваність рекламованих ресурсів може збільшитися. В деяких випадках відвідуваність може впасти, якщо категорія, до якої віднесено рекламне повідомлення є не популярною серед користувачів електронної пошти.

Програмний продукт являє собою веб-додаток електронної пошти для обміну повідомленнями. Для його інтеграції з іншими системами електронної пошти необхідно додати можливість проходити аутентифікацію у системі за допомогою відповідної системи. Такий підхід також надасть можливість інтеграції не лише з поштовими системами, а й з соціальними мережами або месенджерами.

5.4. **Конкурентні переваги рішення**

Оскільки подвійна класифікація повідомлень, а також класифікація за декількома категоріями не використовується у сучасних системах електронної пошти, точного аналогу запропонованого рішення на сьогоднішній день не існує.

Частковим рішенням є використання категорій, віднесення до яких визначається завдяки аналізу метаданих повідомлення. Таке рішення реалізовано у системі Gmail. Ця електронна поштова система надає

можливість розділення повідомлень за базовими категоріями, такими як “Соціальні мережі”, “Оновлення”, “Форуми”, “Реклама”. Крім того, користувач може створювати власні категорії але повідомлення до них він зможе вносити лише вручну. На рис. 5.2 зображена реалізація категорій в системі Gmail.

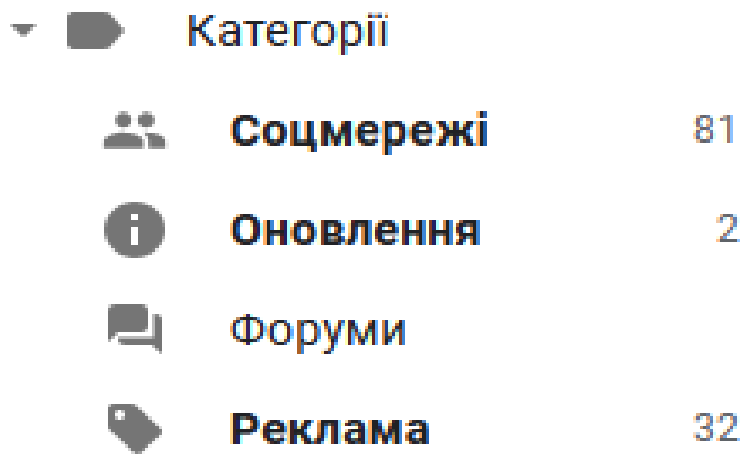


Рис. 5.2. Реалізація категорій в системі Gmail

Недоліком реалізації в системі Gmail є те, що використання метаданих повідомлення досить сильно обмежує кількість категорій, до якого воно може бути віднесено. Крім того, в Gmail повідомлення може бути віднесено лише до однієї категорії.

Інші електронні поштові системи або надають можливість класифікувати повідомлення за метаданими аналогічну реалізованій в Gmail або зовсім не мають можливості класифікувати повідомлення.

Враховуючи всі існуючі недоліки існуючого конкурента, можна визначити основні конкурентні переваги програмного продукту, що пропонується:

1. Підвищена точність класифікації.
2. Можливість класифікації за декількома категоріями.
3. Можливість додавати нові категорії без зміни програмної логіки системи.

5.5. **Клієнти. Сегменти ринку споживання**

Клієнтами програмного продукту, що пропонується є користувачі різних електронних поштових систем.

В світі існує велика кількість поштових систем, які розраховані на різні сегменти користувачів. Зазвичай електронні поштові системи орієнтуються на користувачів, які живуть в одній країні (QQ.com), або спілкуються однією мовою (Mail.ru).

На сьогоднішній день майже кожна система електронної пошти розраховує на англомовний сегмент користувачів. Хоча найпопулярнішою в світі є китайська мова (1.3 мільярди носіїв), англійська мова (600 мільйонів носіїв) є міжнародною, а також вважається найбільш розповсюдженою серед усіх країн.

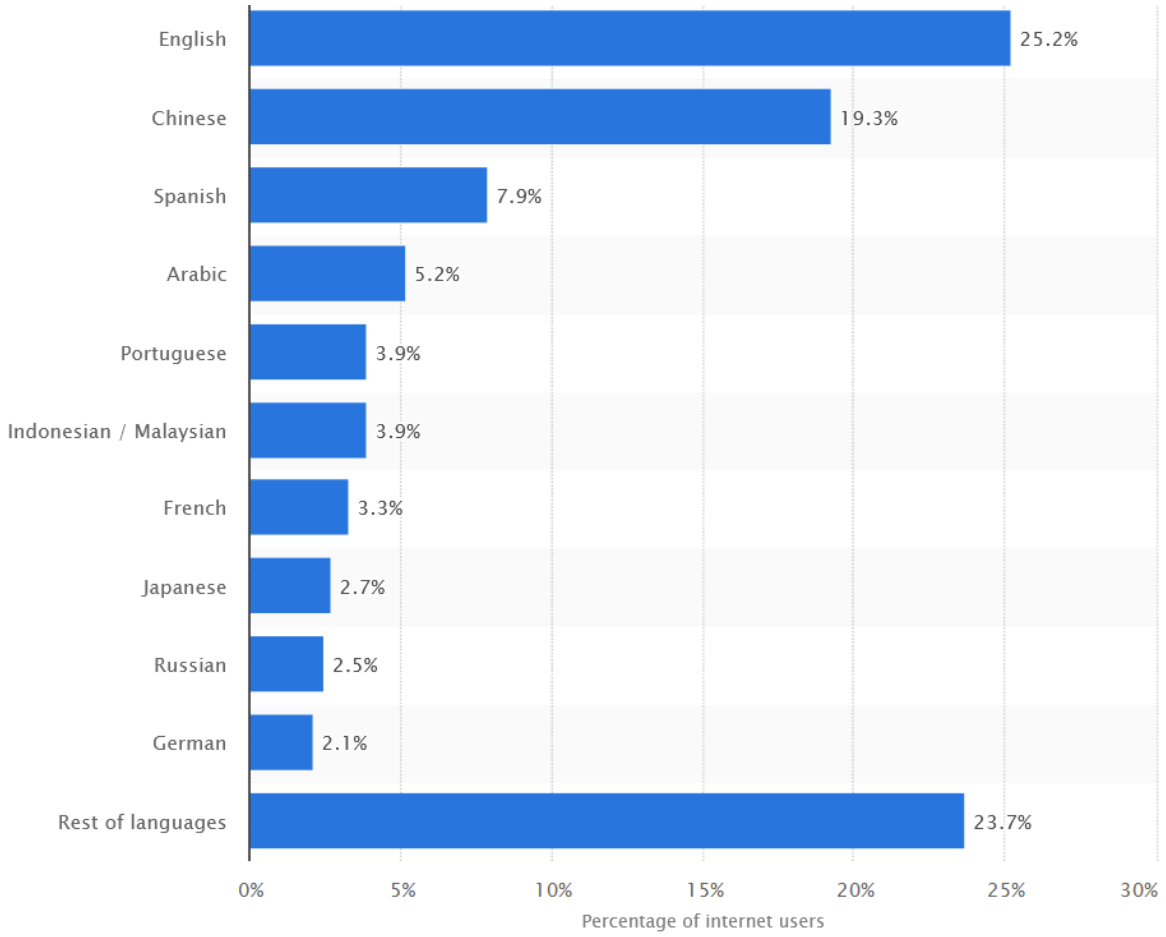


Рис. 5.3. Статистика найпопулярніших мов в Інтернеті

Окрім цього, за статистикою 2019 року англійська мова є найпопулярнішою мовою в Інтернеті (рис. 5.3).

Отже, найбільш перспективним для розробки веб-додатку електронної пошти є англомовний сегмент користувачів, оскільки він є найбільш розповсюдженим в Інтернеті. Для розробки програмного забезпечення відповідно найкраще обрати англійську мову для створення класифікатора та інтерфейсу користувача.

5.6. **Унікальна ціннісна пропозиція**

Ціннісна пропозиція (value proposition) вирішує, чи буде клієнт знайомитися з продуктом. Це коротке пояснення того, як продукт вирішує проблему [15].

Основними проблемами, які були описані в пункті 5.1 є недостатня точність автоматизованої класифікації текстового вмісту повідомлень електронної пошти, а також відсутність класифікації за декількома категоріями.

Були визначені зацікавлені сторони, а також проаналізовані їх інтереси та вплив. Для поштових систем найважливіше – підтримувати свою популярність серед користувачів, адже це основний фактор, який приваблює рекламодавців. Для користувачів поштових систем, в свою чергу, найважливішим є підвищення точності класифікації повідомлень, присутність в системі класифікації за декількома категоріями, а також інтуїтивний інтерфейс. Запропоноване програмне рішення дозволяє задовільнити вимоги користувачів поштових систем в обраному у пункті 5.5 сегменті, а також вирішити основні проблеми.

Враховуючи інформацію з попередніх пунктів цього розділу можна сформувати унікальну ціннісну пропозицію: розроблений веб-додаток, який вирішує проблему неточної класифікації повідомлень завдяки реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

5.7. **Доходи та витрати**

Користуватись веб-додатком зможе будь-який користувач з доступом до Інтернету. Користувач буде мати безкоштовний доступ до базових категорій повідомлень в системі. Для розширення списку категорій користувач зможе оформити платну підписку.

Загалом програми діляться на дві великі групи: вільне програмне забезпечення та пропрієтарне програмне забезпечення [15].

Вільне програмне забезпечення надає користувачу ряд свобод:

1. Запускати програму.
2. Вивчати і змінювати її вихідний код відповідно до власних потреб.
3. Вільно розповсюджувати копії програм.
4. Розповсюджувати модифіковані версії програм.

У випадку, якщо одна з цих свобод відсутня, програма не належить до вільного програмного забезпечення. Тож якщо програмне забезпечення надається безкоштовно, це ще не означає, що воно є вільним. І навпаки, вільне програмне забезпечення може розповсюджуватись за гроші, але не може бути при цьому захищене від розповсюдження копій.

Пропрієтарним є програмне забезпечення, яке при розповсюдженні надає обмежені права на користування ним. До обмежень використання такого програмного забезпечення може відноситись:

1. Закритий доступ до вихідного коду.
2. Заборона на внесення змін.
3. Заборона розповсюдження.
4. Заборона тиражування.
5. Заборона перепродажу.

Для запропонованого веб-додатку планується закриття доступу до програмного коду. Наявність цього пункту відносить розроблений веб-додаток до пропрієтарного програмного забезпечення.

Окрім типу програмного забезпечення необхідно також визначити ліцензію за якою воно буде розповсюджуватись. Існує три основних типи на

які можна поділити програмне забезпечення: безкоштовне, умовно-безкоштовне та комерційне.

Ліцензійна угода безкоштовного програмного забезпечення не потребує ніякої плати його власнику, а також не передбачає додаткових послуг та нових версій. Безкоштовне програмне забезпечення передбачає його безкоштовне використання протягом необмеженого терміну, а також без обмежень у функціональності.

До умовно-безкоштовного відноситься програмне забезпечення, яке розповсюджується безкоштовно, але при цьому користувачу надається не повна версія комерційного продукту, яка є обмежена за функціональними можливостями, терміном дії або дозволом на використання лише в ознайомчих цілях.

Якщо програмне забезпечення створене для отримання прибутку, воно називається комерційним. Для таких продуктів зазвичай діє безкоштовна підтримка протягом певного терміну після випуску.

Для запропонованого веб-додатку планується його розповсюдження на умовно-безкоштовній основі з варіантами місячної або річної оплати підписки, яка розширює список категорій за якими можна групувати повідомлення електронної пошти.

Для визначення сумарного доходу від запропонованого веб-додатку необхідно порахувати витрати на його розробку та підтримку, а також прибутки за продаж платної підписки для розширення кількості категорій за якими можна групувати повідомлення.

Перед підрахунком витрат необхідно врахувати кількість людей в команді. Для розробки запропонованого веб-додатку необхідно набрати команду з 6 працівників, що є оптимальною кількістю людей для виконання наявного об’єму задач.

На розробку та супровід програмного продукту планується витратити в загальному обсязі 6 місяців. Після розробки програмного продукту планується витратити на його супровід від 6 місяців. Після закінчення

розробки програмного продукту для подальшого супроводу доцільне скорочення команди до 4 учасників.

До списку необхідних витрат для розробки та супроводу продукту відносяться:

1. Виплата заробітних плат учасникам команди – в середньому $1500 в місяць на людину.
2. Оренда невеликого приватного офісу – $200 в місяць за 1 робоче місце.
3. Покупка обладнання – 6 ноутбуків/комп’ютерів, орієнтовно $1000 за одиницю.
4. Маркетинг – $1200 в місяць протягом 4 місяців на рекламні тарифи Google Ads та Facebook Ads.
5. Послуги юриста – $700 в місяць.
6. Послуги бухгалтера – $600 в місяць.
7. Послуги прибиральника – $200 в місяць.
8. Податки.

На кінець другого півріччя за припущенням сумарний дохід за виключенням податків повинен складати $71400.

Ціна придбання місячної підписки становить $5, а річної – 50$. За перший місяць планується продати 2000 місячних та 200 річних підписок. За допомогою реклами Google Ads та Facebook Ads після завершення розробки програмного продукту планується залучати нових користувачів та збільшувати кількість проданих ліцензій за наступні місяці.

Оскільки перше півріччя буде витрачено на розробку технічних вимог, архітектури, а також клієнтської та серверної частини веб-додатку, очікується що на повний цикл розробки необхідно буде витратити 72000 доларів, з яких 54000 буде витрачено на зарплати, 7200 на оренду приміщень, 6000 на обладнання, 4200 на послуги юриста, 3600 на послуги бухгалтера і 1200 на послуги прибиральниці. В табл. 5.2 зображені детальні витрати першого півріччя.

Таблиця 5.2

Витрати та прибутки за перше півріччя

	№ Місяця						Підсумок
Прибутки / Витрати (тис. дол.)	1	2	3	4	5	6	Перше півріччя
Продаж підписок	-	-	-	-	-	-	-
Зарплата	-9	-9	-9	-9	-9	-9	-54
Оренда	-1.2	-1.2	-1.2	-1.2	-1.2.	-1.2	-7.2
Обладнання	-6	-	-	-	-	-	-6
Маркетинг	-	-	-	-	-	-	-
Юрист	-0.7.	-0.7	-0.7	-0.7	-0.7	-0.7	-4.2
Бухгалтер	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-3.6
Прибирання	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-1.2
Прибутки – витрати	-11.7	-11.7	-11.7	-11.7	-11.7	-11.7	-76.2

Починаючи з другого півріччя планується старт рекламної кампанії, а також продаж платних підписок для розширення списку категорій за якими користувач зможе групувати повідомлення. За друге півріччя планується отримати прибуток з продажу підписок у розмірі 198000 доларів. Запланований чистий прибуток після закінчення другого півріччя повинен становити 71400 доларів. В табл. 5.3 зображені детальні прибутки та витрати першого та другого півріччя.

Таблиця 5.3

Витрати та прибутки за друге півріччя

	№ Місяця						Підсумок
Прибутки / Витрати (тис. дол.)	1	2	3	4	5	6	Друге півріччя
Продаж підписок	20	30	35	38	40	35	198
Зарплата	-6	-6	-6	-6	-6	-6	-90
Оренда	-0.8	-0.8	-0.8	-0.8	-0.8	-0.8	-12
Обладнання	-	-	-	-	-	-	-6
Маркетинг	-1.2	-1.2	-1.2	-1.2	-	-	-4.8
Юрист	-	-	-	-	-	-	-4.2
Бухгалтер	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-7.2
Прибирання	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-2.4
Прибутки – витрати	11.2	21.2	26.2	29.2	32.4	27.4	71.4

5.8. **Бізнес модель**

Для створення бізнес-моделі необхідно узагальнити інформацію всіх попередніх пунктів та перенести її на канву (рис. 5.4).

Проблема: недостатня точність автоматизованої класифікації текстового вмісту повідомлень електронної пошти, а також відсутність класифікації за декількома категоріями.

Рішення: веб-додаток електронної пошти з можливістю групувати повідомлення за категоріями.

Унікальна ціннісна пропозиція: розроблений веб-додаток, який вирішує проблему неточної класифікації повідомлень завдяки реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Споживачі: англомовний сегмент користувачів електронної пошти.

Потоки доходів: продаж місячної та річної підписки на розширення списку категорій.

Структура витрат: заробітні плати учасникам команди, юристу, бухгалтеру та прибиральнику, оренда приміщення, обладнання, маркетинг, податки.

Канали збуту: власний магазин в веб-додатку.

Ключові партнери: компанії-власники електронних поштових систем.

Ключові активності: створення документації, розробка програмного продукту, маркетинг, супровід програмного продукту.

Ключові метрики: кількість проданих підписок, відгуки користувачів.

<table>
<tr><td rowspan="2">Проблема
недостатня точність класифікації;
відсутність класифікації за декількома категоріями;
недосконалість існуючих рішень.</td><td>Рішення
веб-додаток електронної пошти з можливістю групувати повідомлення за категоріями.</td><td rowspan="2" colspan="2">Унікальна ціннісна пропозиція
розроблений веб-додаток, який вирішує проблему неточної класифікації повідомлень завдяки реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти.</td><td rowspan="2">Ключові активності
створення документації;
розробка програмного продукту;
маркетинг;
супровід програмного продукту</td></tr>
<tr><td>Ключові метрики:
кількість проданих підписок
відгуки користувачів</td></tr>
<tr><td rowspan="3">Структура витрат
заробітні плати учасникам команди, юристу, бухгалтеру та прибиральнику;
аренда приміщення, обладнання;
маркетинг;
податки.</td><td>Ключові партнери
компанії-власники електронних поштових систем</td><td>Канали збуту
Власний магазин в веб додатку</td><td colspan="2">Споживачі
англомовний сегмент користувачів електронних поштових систем</td></tr>
<tr><td colspan="4">Потоки доходів
продаж місячної та річної підписки на розширення списку категорій.</td></tr>
</table>

Рис. 5.4. Канва бізнес-моделі

5.9. **Висновки до розділу 5**

В даному розділі розглянуто сферу електронних поштових систем, визначено основні проблеми та побудовано відповідне дерево проблем.

Визначено та проаналізовано зацікавлені сторони, виділено їх інтереси, вплив, а також стратегію приваблення. За наявними проблемами сформовано комерційне рішення, а саме веб-додаток електронної пошти з можливістю групування повідомлень за категоріями.

Проаналізовані основні аналоги, визначені їх недоліки, а також переваги запропонованого рішення.

Крім цього, визначено майбутніх клієнтів, а також проаналізовано існуючи сегменти ринку споживання. Сформовано унікальну ціннісну пропозицію програмного рішення, розроблений веб-додаток, який вирішує проблему неточної класифікації повідомлень завдяки реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

Також визначені основні потоки доходів, структура витрат та пораховані витрати на розробку і супровід програмного продукту протягом року, а також можливі доходи з продажу місячних та річних підписок на розширення списку категорій.

Сформовано канву бізнес-моделі, для представлення структурних, операційних та фінансових механізмів роботи для створення запропонованого веб-додатку.

ВИСНОВКИ

В даній роботі запропоновано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти з метою підвищення точності класифікації, а також надання можливості класифікації повідомлення за декількома категоріями.

Проаналізовано існуючі методи класифікації текстового вмісту повідомлень, визначено їх основні переваги та недоліки для класифікації повідомлень електронної пошти.

За результатами виконаного аналізу існуючих методів запропоновано розділення етапів фільтрації та класифікації повідомлень за категоріями, а також використання методу опорних векторів на етапі фільтрації і методу Баєса на етапі класифікації. Також запропоновано модифікацію метода Баєса, а саме введення значення граничної ймовірності, а також використання логарифмічної оцінки для підвищення точності та класифікації текстового вмісту повідомлень електронної пошти.

Запропонований метод був реалізований у вигляді веб-додатку електронної пошти з можливістю групування повідомлень за категоріями. Обгрунтований вибір технологій, обраних для розробки веб-додатку. Описана архітектура розробленого веб-додатку, а також структура його серверної та клієнтської частин.

Проведено порівняння результатів класифікації комбінованим методом автоматизованої класифікації текстового вмісту повідомлень електронної пошти з іншими методами машинного навчання. Визначено, що реалізований метод надає більшу точність класифікації повідомлень ніж інші методи, використані для порівняння.

Виконаний аналіз економічної конкурентоспроможності реалізованого методу. За результатами аналізу сформовані таблиці прогнозованих витрат та прибутків від розробленого веб-додатку за перші дванадцять місяців, а також побудовано бізнес-модель.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Email Statistics Report, 2015-2019. [Електронний ресурс]. – Режим доступу: https://bit.ly/2PvB971. Дата доступу – Жовтень 2019.
2. О задачах классификации. [Електронний ресурс]. – Режим доступу: http://bazhenov.me/blog/2012/06/05/classification.html. Дата доступу – Жовтень 2019.
3. Muller, Andreas. Introduction to Machine Learning with Python / A. C. Muller, S. Guido. – O'Reilly Media, Inc., 2017. – 400 p. – ISBN: 978-1449369415.
4. Downey, Allen. Think Bayes: Bayesian Statistics in Python / A. B. Downey. – O'Reilly Media, Inc., 2017. – 214 p. – ISBN: 978-1449369415.
5. Воронцов, Константин. Лекции по методу опорных векторов : статья / К. В. Воронцов. – Москва : ВЦ РАН, 2017. – 18 с.
6. Conway, Drew. Machine Learning for Email: Spam Filtering and Priority Inbox / D. Conway, J. M. White. – O'Reilly Media, Inc., 2012 – 148 p. – ISBN: 978-1449314309.
7. Hartshorn, Scott. Machine Learning With Random Forests And Decision Trees: A Visual Guide For Beginners / S. Hartshorn. 2016. – 74 p. – ASIN: B01JBL8YVK.
8. How to solve 90% of NLP problems: a step-by-step guide [Електронний ресурс]. / Emmanuel Ameisen – Режим доступу : https://bit.ly/2seCcjA. Дата доступу – Жовтень 2019.
9. Nielsen, Michael. Neural Networks and Deep Learning / M. A. Nielsen – Determination Press, 2015.
10. Сузи, Р. А. Язык программирования Python: Учебное пособие / Р. А. Сузи – М. : Интуит, Бином. Лаборатория знаний, 2006. – 328 с.
11. Добро пожаловать во Flask [Електронний ресурс]. – Режим доступу : http://flask-russian-docs.readthedocs.io/ru/latest/. Дата доступу – Вересень 2019

12. TypeScript Documentation [Електронний ресурс]. – Режим доступу : https://www.typescriptlang.org/docs. Дата доступу – Вересень 2019.
13. Tour of Heroes App and Tutorial [Електронний ресурс]. – Режим доступу : https://angular.io/tutorial. Дата доступу – Вересень 2019.
14. Getting Started with Angular Material [Електронний ресурс]. – Режим доступу : https://material.angular.io/guide/getting-started. Дата доступу – Вересень 2019.
15. Барасюк, Ярослав. Інформаційні системи і технології в економіці : курс лекцій / Я. М. Барасюк, О. В. Стець – Київ: Київський національний торгово-економічний університет. – 398 с.
16. Most common languages used on the internet [Електронний ресурс]. – Режим доступу : https://bit.ly/2YB3eOr. Дата доступу – Листопад 2019.

ДОДАТКИ

Додаток 1

Лістинг програми

Лістинг 1. app.py

```
from flask import Flask, send_from_directory
from apps import APPS
from database import set_db


def _setup_blueprints(main_app, app_list):
    for app in app_list:
        app.setup_urls()
        app.setup_controllers()
        app.setup_models()
        main_app.register_blueprint(app, url_prefix=f'/api/{app.name}')


def create_app(config):
    main_app = Flask(__name__)
    main_app.config.from_object(config)
    _setup_blueprints(main_app, APPS)
    main_app.app_context().push()
    set_db(main_app)

    return main_app


if __name__ == '__main__':
    from config import DebugConfig
    app = create_app(DebugConfig)
    # only for development
    from flask_cors import CORS
    CORS(app)


    @app.route('/static/<path:path>')
    def serve_static(path):
        return send_from_directory('static', path)

    app.run()
```

Лістинг 2. base_lassifier.py

```
import csv, pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from .dataset_pre_processor import DatasetPreProcessor


class BaseClassifier:

    def __init__(self, dataset_dir, classifier, message_type):
        self.dataset_dir = dataset_dir
        self.type = message_type
        self.messages_dir =
f'{self.dataset_dir}/{self.type}/{self.type}_messages.csv'
        self.labels_dir =
f'{self.dataset_dir}/{self.type}/{self.type}_labels.csv'
```

```
        self.vocab_dir =
f'{self.dataset_dir}/{self.type}/{self.type}_vocab.json'
        self.model_dir =
f'helper_classes/message_classifier/models/{self.type}_model.sav'
        self.vectorizer_dir =
f'helper_classes/message_classifier/models/{self.type}_vectorizer.sav'
        self.clf = classifier
        self.model_loaded = False

        self.vectorizer = CountVectorizer()
        self.X_train, self.X_test, self.y_train, self.y_test = None, None,
None, None

    @staticmethod
    def _read_messages(filename):
        with open(filename, 'r') as file:
            messages = []
            reader = csv.reader(file)
            for row in reader:
                messages.append(' '.join(row))
            return messages

    @staticmethod
    def _read_labels(filename):
        with open(filename, 'r') as file:
            reader = csv.reader(file)
            labels = next(reader)
            return labels

    def classify_message(self, message):
        if not self.model_loaded:
            self.load_data()
        message = '
'.join(DatasetPreProcessor.process_message_text(message))
        vector = self.vectorizer.transform([message]).toarray()
        return self.clf.predict(vector)

    def _vectorize_messages(self, messages):
        X = self.vectorizer.fit_transform(messages)
        return X.toarray()

    def _save_data(self):
        pickle.dump(self.clf, open(self.model_dir, 'wb'))
        pickle.dump(self.vectorizer, open(self.vectorizer_dir, 'wb'))

    def load_data(self):
        try:
            self.clf = pickle.load(open(self.model_dir, 'rb'))
            self.vectorizer = pickle.load(open(self.vectorizer_dir, 'rb'))
            self._initialize_data()
            self.model_loaded = True
        except:
            self.train_classifier()

    def get_accuracy(self):
        if not self.model_loaded:
            self.load_data()
        result = self.clf.score(X=self.X_test, y=self.y_test)
        return result

    def _split_dataset(self, X, y):
```

```
        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(X, y, test_size=0.33, random_state=42)

    def _initialize_data(self):
        messages = self._read_messages(self.messages_dir)
        labels = self._read_labels(self.labels_dir)
        message_vectors = self._vectorize_messages(messages)
        self._split_dataset(message_vectors, labels)

    def train_classifier(self):
        self._initialize_data()
        self.clf.fit(X=self.X_train, y=self.y_train)
        self.model_loaded = True
        self._save_data()
```

Лістинг 3. spam_classifier.py

```
from .base_classifier import BaseClassifier
from sklearn import svm
import time


class SpamClasifier(BaseClassifier):

    def __init__(self):
        SVM_classifier = svm.SVC()

super().__init__('helper_classes/message_classifier/datasets/ready',
SVM_classifier, 'spam')
```

Лістинг 4. categories_classifier.py

```
from .base_classifier import BaseClassifier
from sklearn import naive_bayes
from .dataset_pre_processor import DatasetPreProcessor
from nltk import sent_tokenize
from collections import Counter
import time



class CategoriesClassifier(BaseClassifier):

    THRESHOLD_VALUE = 0.8

    def __init__(self):
        Bayes_classifier = naive_bayes.BernoulliNB()

super().__init__('helper_classes/message_classifier/datasets/ready',
Bayes_classifier, 'categories')

    def get_sentence_labels(self, sentence):
        if not self.model_loaded:
            self.load_data()
```

```
        sentence = '
'.join(DatasetPreProcessor.process_message_text(sentence))
        vector = self.vectorizer.transform([sentence]).toarray()
        probabilities = self.clf.predict_proba(vector)
        result = []
        for i, proba in enumerate(probabilities[0]):
            if proba >= self.THRESHOLD_VALUE:
                result.append(i)

        return result

    def get_message_labels(self, message):
        sentences = sent_tokenize(message)
        categories = []
        for sentence in sentences:
            categories.extend(self.get_sentence_labels(sentence))
        result = []
        categories_dict = Counter(categories)
        for category, amount in categories_dict.items():
            if amount / len(categories) > 0.3:
                result.append(category)
        return result if result != [] else ['4']
```

Лістинг 5. data_pre_processor.py

```
import nltk, re, os, csv, json, itertools
from nltk.corpus import stopwords
from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction import DictVectorizer
# import sys
# sys.path.append("./api/helper_classes/message_classifier")


class DatasetPreProcessor:

    spam_vocab = {'Spam': 0, 'Not Spam': 1}
    categories_vocab = None
    spam_messages = None
    categories_messages = None
    spam_labels = None
    categories_labels = None

    output_dir = 'datasets/ready'
    spam_dir = 'spam'
    categories_dir = 'categories'
    raw_dataset_dir = 'helper_classes/message_classifier/datasets/raw'
    output_dir = 'helper_classes/message_classifier/datasets/ready'

    def __init__(self):
        self.spam_messages_dir =
f'{self.raw_dataset_dir}/spam/spam_dataset'
        self.spam_labels_dir =
f'{self.raw_dataset_dir}/{self.spam_dir}/spam-labels.tr.label'
        self.categories_dataset_dir =
f'{self.raw_dataset_dir}/{self.categories_dir}/categories_dataset.json'

    def process_spam_dataset(self):
```

```
        files = self._get_spam_files()
        self.spam_labels = self._get_spam_labels()
        self.spam_messages = self._process_spam_files(files)

    def process_categorized_dataset(self):
        data = self._get_categorized_data()
        self.categories_messages, self.categories_labels,
self.categories_vocab \
            = self._process_categorized_data(data)

    def save_results(self):
        spam_dir = f'{self.output_dir}/{self.spam_dir}'
        categories_dir = f'{self.output_dir}/{self.categories_dir}'

        self._save(self.spam_messages, f'{spam_dir}/spam_messages.csv')
        self._save(self.spam_labels, f'{spam_dir}/spam_labels.csv')
        self._save_json(self.spam_vocab, f'{spam_dir}/spam_vocab.json')

        self._save(self.categories_messages,
f'{categories_dir}/categories_messages.csv')
        self._save(self.categories_labels,
f'{categories_dir}/categories_labels.csv')
        self._save_json(self.categories_vocab,
f'{categories_dir}/categories_vocab.json')

    def _get_spam_files(self):
        files = os.listdir(self.spam_messages_dir)
        # Sort files by number in name
        files.sort(key=lambda x: int(x[6:-4]))
        return files

    def _get_spam_labels(self):
        with open(self.spam_labels_dir, 'r') as file:
            raw_labels = file.readlines()
        labels = self._parse_labels(raw_labels)
        return labels

    @staticmethod
    def _parse_labels(raw_labels):
        labels = []
        for row in raw_labels[1:]:
            labels.append(row.strip().split(',')[1])
        return labels

    def _process_spam_files(self, files):
        message_list = []
        for f in files:
            text = self._get_file_text(f'{self.spam_messages_dir}/{f}')
            message = self.process_message_text(text)
            message_list.append(message)

        return message_list

    @staticmethod
    def _cleanhtml(raw_html):
        cleanr = re.compile(r'<[\s\S]*?>')
        cleantext = re.sub(cleanr, '', raw_html)
        return cleantext

    @staticmethod
    def _get_file_text(filename):
```

```
        with open(filename, 'r', encoding='utf8', errors='ignore') as file:
            return file.read()

    @classmethod
    def process_message_text(cls, text):
        # Remove html tags
        text = cls._cleanhtml(text)

        text.encode("ascii", errors="ignore").decode()

        # Split words
        tokens = word_tokenize(text)

        # Only alphabetic symbols
        words = [word for word in tokens if word.isalpha()]

        # Lowercase
        words = [word.lower() for word in words]

        # Remove stopwords
        stop_words = set(stopwords.words('english'))
        words = [w for w in words if not w in stop_words]

        # Stem words
        porter = PorterStemmer()
        words = [porter.stem(word) for word in words]
        return words

    @classmethod
    def _save(cls, data, filename):
        with open(filename, 'w', newline='') as file:
            writer = csv.writer(file)
            if isinstance(data[0], list):
                writer.writerows(data)
            else:
                writer.writerow(data)

    @classmethod
    def _save_json(cls, data, filename):
        with open(filename, 'w', newline='') as file:
            json.dump(data, file)

    def _get_categorized_data(self):
        data = []
        with open(self.categories_dataset_dir, 'r') as file:
            for line in file:
                data.append(json.loads(line))
        return data

    @classmethod
    def _create_categories_vocabulary(cls, labels):
        categories = list(set(labels))
        vocab = dict(zip(categories, [x for x in range(0,
len(categories))]))
        return vocab

    @classmethod
    def _vectorize_labels(cls, labels):
        vocab = cls._create_categories_vocabulary(labels)
        labels = [vocab[lable] for lable in labels]
        return labels, vocab
```

```
    @classmethod
    def _process_categorized_data(cls, data):
        messages = []
        labels = []
        for message in data:
            content = message['content']
            category = message['annotation']['label'][0]
            message = cls.process_message_text(content)
            messages.append(message)
            labels.append(category)

        labels, vocab = cls._vectorize_labels(labels)
        return messages, labels, vocab
```

Лістинг 6. Mail model - view.py

```
from helper_classes.base_view import BaseView
from flask import current_app, request, g
from .schemas.message_schema import MessageSchema
from marshmallow import ValidationError
from werkzeug.exceptions import Unauthorized
from helper_classes.auth_decorator import login_required


class MessageView(BaseView):
    def __init__(self):
        self.message_controller =
current_app.blueprints['mail'].controllers.MessageController

    @login_required
    def post(self):
        try:
            message_data = MessageSchema().load(request.json)
        except ValidationError as e:
            return self._get_response(e.messages, status_code=400)
        data, status_code =
self.message_controller.create_message(message_data)
        return self._get_response(data, status_code=status_code)

    @login_required
    def get(self):
        response, status_code = self.message_controller.get_user_messages()
        return self._get_response(response, status_code=status_code)

class TypesView(BaseView):
    def __init__(self):
        self.message_controller =
current_app.blueprints['mail'].controllers.MessageController

    @login_required
    def get(self):
        response, status_code = self.message_controller.get_types()
        return self._get_response(response, status_code=status_code)
```

Лістинг 7. message_controller.py

```
from flask import current_app, g
from helper_classes.auth_decorator import login_required
from marshmallow import ValidationError
from helper_classes.message_classifier.spam_classifier import SpamClasifier
from helper_classes.message_classifier.categories_classifier import
CategoriesClassifier
import uuid


class MessageController:

    @classmethod
    def create_message(cls, data):
        sender = current_app.models.User.get_user_by_id(g.user_id)
        receiver =
current_app.models.User.get_user_by_email(data.pop('r_email'))
        data['sender'] = sender
        data['receiver'] = receiver
        if cls._message_is_spam(data.get('message', '')):
            data['is_spam'] = True
        else:
            data['message_types'] =
cls._get_message_types(data.get('message'))
        current_app.models.Message.create_message(data)
        return 'Message was sended', 200

    @classmethod
    def get_user_messages(cls):
        sent_messages =
current_app.models.Message.get_sent_messages_by_user_id(g.user_id)
        received_messages =
current_app.models.Message.get_received_messages_by_user_id(g.user_id)
        result = {}

        for message in sent_messages:
            result.setdefault('Sent', []).append(message.get_public_data())
        for message in received_messages:
            if message.is_spam:
                result.setdefault('Spam',
[]).append(message.get_public_data())
                continue
            result.setdefault('All', []).append(message.get_public_data())

            for message_type in message.message_types:
                result.setdefault(message_type.name,
[]).append(message.get_public_data())
        return result, 200

    @staticmethod
    def _get_message_types(message):
        categories_classifier = CategoriesClassifier()
        raw_types = categories_classifier.get_message_labels(message)
        types = []

        for t in raw_types:
            id = int(t)
            types.append(current_app.models.Type.get_type_by_id(id))
        # types = current_app.models.Type.get_all_types()
```

```
        # types = [current_app.models.Type.get_type_by_name('Unsorted')]
        #
types.append(current_app.models.Type.get_type_by_name('Business'))

        return types

    @staticmethod
    def _message_is_spam(message):
        spam_classifier = SpamClasifier()
        result = spam_classifier.classify_message(message)
        return result[0] == '0'


    @classmethod
    def get_types(cls):
        types = current_app.models.Type.get_all_types()
        return types, 200
```

Лістинг 8. message_model.py

```
from database import db

message_type_table = db.Table('message_type',
    db.Column('message_id', db.Integer,
db.ForeignKey('message.message_id'), primary_key=True),
    db.Column('type_id', db.Integer, db.ForeignKey('type.type_id'),
primary_key=True)
)

class Message(db.Model):
    __tablename__ = 'message'
    __table_args__ = {'extend_existing': True}
    message_id = db.Column(db.Integer, primary_key=True)
    subject = db.Column(db.String(50))
    message = db.Column(db.String(10000))
    date = db.Column(db.DateTime())
    is_read = db.Column(db.Boolean, nullable=False, default=False)
    is_spam = db.Column(db.Boolean, nullable=False, default=False)
    sender_id = db.Column(db.Integer,
db.ForeignKey('user_profile.user_id'), nullable=False)
    sender = db.relationship('User', foreign_keys=sender_id,
backref='sended_messages')
    receiver_id = db.Column(db.Integer,
db.ForeignKey('user_profile.user_id'))
    receiver = db.relationship('User',foreign_keys=receiver_id,
backref='received_messages')
    message_types = db.relationship('Type', secondary=message_type_table,
lazy=True,
                                    backref=db.backref('messages',
lazy=True))


    @classmethod
    def create_message(cls, data):
        message = cls(**data)
        db.session.add(message)
        db.session.commit()
        return message
```

```
    @classmethod
    def get_sent_messages_by_user_id(cls, user_id):
        return
cls.query.filter(cls.sender_id==user_id).order_by(cls.date.desc())

    @classmethod
    def get_received_messages_by_user_id(cls, user_id):
        return
cls.query.filter(cls.receiver_id==user_id).order_by(cls.date.desc())

    def get_public_data(self):
        if self.receiver is None:
            receiver = ' '
        else:
            receiver = self.receiver
        public_data = {
            "message_id": self.message_id,
            "subject": self.subject,
            "message": self.message,
            "date": self.date,
            "is_read": self.is_read,
            "is_spam": self.is_spam,
            "sender": self.sender,
            "receiver": receiver,
            "types": self.message_types,
        }
        return public_data

    def __repr__(self):
        return f'<Message {self.subject}>'
```

Лістинг 9. type_model.py

```
from database import db


class Type(db.Model):
    __tablename__ = 'type'
    __table_args__ = {'extend_existing': True}
    type_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    # messages = db.relationship('Message', lazy=True,
    #     cascade='all, delete, delete-orphan', backref='message_type')

    @classmethod
    def create_type(cls, data):
        t = cls(name=data)
        db.session.add(t)
        db.session.commit()
        return t

    @classmethod
    def get_all_types(cls):
        return cls.query.all()

    @classmethod
    def get_type_by_name(cls, name):
        return cls.query.filter(cls.name == name).first()
    @classmethod
```

```
    def get_type_by_id(cls, id):
        return cls.query.filter(cls.type_id == id).first()


    def __repr__(self):
        return f'<Type {self.name}>'
```

Лістинг 10. User module - view.py

```
import redis
from flask import current_app, request, g, send_from_directory
from .schemas.UserRegisterSchema import UserRegisterSchema
from marshmallow import ValidationError
from apps.user_app.schemas.login_schema import LoginSchema
from apps.user_app.schemas.social_login_schema import SocialLoginSchema
from apps.user_app.schemas.update_user_schema import UpdateUserSchema
from helper_classes.base_view import BaseView
import facebook
from helper_classes.auth_decorator import login_required
from helper_classes.handy_functions import try_except


class UserView(BaseView):

    def __init__(self):
        self.user_controller =
current_app.blueprints['user'].controllers.UserController

    def post(self):
        # tofix
        request_data = request.json
        try:
            UserRegisterSchema().load(request_data)
        except ValidationError as err:
            return self._get_response(data=err.messages, status_code=409)
        data = [
            self.user_controller.register_user(**request_data),
        ]
        return self._get_response(data, status_code=201)

    @login_required
    def get(self):
        user_data = self.user_controller.get_profile(user_id=g.user_id)
        return self._get_response(data=user_data)

    @login_required
    def patch(self):
        try:
            data = UpdateUserSchema().load(data=request.json)
            if data.get('new_password', None):
                message, status_code =
self.user_controller.change_password(user_id=g.user_id, **data)
            else:
                message, status_code =
self.user_controller.update_user(user_id=g.user_id, data=data)
        except ValidationError as e:
            return self._getrol_response(e.messages, status_code=400)
        return self._get_response(data=message, status_code=status_code)
```

```
    @login_required
    def put(self):
        user_data = request.json
        name = user_data['name']
        surname = user_data['surname']
        user_profile_controller =
current_app.blueprints['user'].controllers.UserController
        user_profile_controller.change_user_data(user_id=g.user_id,
name=name, surname=surname)
        return self._get_response(f'User`s data updated', status_code=200)


class UserAvatarView(BaseView):

    def __init__(self):
        self.user_profile_controller =
current_app.blueprints['user'].controllers.UserController

    @login_required
    def post(self):
        try:
            if 'file' not in request.files:
                return self._get_response("No file part", status_code=422)
            file = request.files['file']

self.user_profile_controller.save_user_avatar(user_id=g.user_id,
avatar=file)
        except ValidationError as e:
            return self._get_response(e.messages, status_code=400)
        return self._get_response(f'User`s avatar updated',
status_code=200)

    def get(self):
        avatar = request.args.get('avatar')
        try:
            return
send_from_directory(self.user_profile_controller.get_user_avatar_path(),
filename=avatar, as_attachment=True)
        except FileNotFoundError as e:
            return self._get_response("file not found", status_code=400)

class LoginView(BaseView):
    def __init__(self):
        self.login_controller =
current_app.blueprints['user'].controllers.LoginController

    def post(self):
        user_data = try_except(LoginSchema().load,
SocialLoginSchema().load, request.json)
        if not user_data:
            return self._get_response('Invalid user data', status_code=400)
        if user_data.get('provider'):
            session_id, user_id =
self.login_controller.login_with_social(data=user_data)
        else:
            session_id, user_id =
self.login_controller.login(data=user_data)
        return self._get_response({"session_id": session_id, "user_id":
user_id}, status_code=201)

class LogoutView(BaseView):
```

```
    @login_required
    def post(self):
        with redis.Redis() as redis_client:
            redis_client.delete(g.user_id)
            redis_client.delete(request.headers.get('Authorization'))
        return self._get_response(data={'message': 'You successfully logged
out.'})
```

Лістинг 11. user_controller.py

```
from helper_classes.email_builder.build_email import build_email
from marshmallow import ValidationError
from flask import current_app, g
import celery
import os
import uuid
from pathlib import Path


class UserController:
    @staticmethod
    def _get_user(user_id):
        user = current_app.models.User.get_user_by_id(user_id)
        return user

    @classmethod
    def register_user(cls, name, email, password, surname=None):

        user = current_app.models.User.get_user_by_email(email=email)

        if user is None:
            user = current_app.models.User.create_user(
                name=name, email=email,
                password=password, surname=surname,

avatar='http://localhost:5000/static/images/user_avatar.png'
            )
            cls.setup_registration_otc(user)
            return 'user created'

        if user.is_active:
            return 'User is already registered'
        if user.is_uuid_valid():
            return 'uuid is valid'

        user.delete_user()
        # tofix
        user = current_app.models.User.create_user(
            name=name, email=email,
            password=password, surname=surname
        )
        cls.setup_registration_otc(user)
        return 'user uuid updated'


    @classmethod
    def activate_user(cls, user_uuid):
        user = current_app.models.User.get_user_by_uuid(user_uuid)
```

```
        if user.is_active:
            return 'user already activated', 409
        if user.is_uuid_valid():
            user.activate_user()
            return 'user activated', 200
        return 'uuid outdated', 409

    @classmethod
    def update_user(cls, user_id, data):
        current_app.models.User.update_user(data, user_id=user_id)
        return 'User was successfully updated', 200

    @classmethod
    def setup_registration_otc(cls, user):
        celery_app = celery.Celery(
            current_app.config['CELERY_APP_NAME'],
            broker=current_app.config['CELERY_BROKER_URL']
        )
        uuid = current_app.blueprints['otc'].controllers.\
            OtcController.create_registration_uuid()
        user.set_uuid(uuid)
        em_type = 'email_confirmation'
        content = {'username': user.name, 'uuid': uuid}
        email_data = build_email(user.email, em_type, **content)
        celery_app.send_task('app.async_email', kwargs = email_data)

    @staticmethod
    def get_profile(user_id):
        user = current_app.models.User.get_user_by_id(user_id=user_id)
        return user.get_public_data()

    @classmethod
    def change_password(cls, user_id, new_password, old_password=None):
        user = current_app.models.User.get_user_by_id(user_id=user_id)
        if (not user.password_is_set()) or (old_password and
user.check_password(old_password)):
            user.set_password(new_password)
            return 'Your password was updated', 200
        else:
            return 'Wrong password', 400

    @classmethod
    def change_user_data(cls, user_id, name, surname):
        user = current_app.models.User.get_user_by_id(user_id=user_id)
        user.change_public_fields(name, surname)

    @classmethod
    def save_user_avatar(cls, user_id, avatar):
        allowed_extensions = {'png', 'jpg', 'jpeg'}
        image_store_url = 'http://localhost:5000/api/user/v1/user/avatar'
        user = current_app.models.User.get_user_by_id(user_id=user_id)
        prefix = uuid.uuid4()
        file_format = avatar.filename[avatar.filename.rindex('.')+1:]
        current_folder = Path().absolute()
        full_path = os.path.join(str(current_folder) +
current_app.config['UPLOAD_FOLDER'])
        if not os.path.isdir(full_path):
            os.makedirs(full_path)
        if file_format in allowed_extensions:
            old_user_avatar_url = user.avatar
```

```
            avatar_file_name = "{}-{}.{}".format(prefix, user_id,
file_format)
            avatar.save(full_path+avatar_file_name)
            user.change_avatar_url('{}?avatar={}'.format(image_store_url,
avatar_file_name))
            if old_user_avatar_url.find('?') != -1:
                old_avatar_name =
old_user_avatar_url[old_user_avatar_url.rindex('=') + 1:]
                cls.delete_old_avatar_file(old_avatar_name)
        else:
            raise ValidationError('Wrong avatar extension')

    @staticmethod
    def delete_old_avatar_file(avatar_file_name):
        full_path =
os.path.join(str(Path().absolute())+current_app.config['UPLOAD_FOLDER']+ava
tar_file_name)
        if os.path.isfile(full_path):
            os.remove(full_path)

    @staticmethod
    def get_user_avatar_path():
        full_path = os.path.join(str(Path().absolute()) +
current_app.config['UPLOAD_FOLDER'])
        return full_path
```

Лістинг 12. login_controller.py

```
import uuid, time, json, redis, facebook
from marshmallow import ValidationError
from flask import current_app
from google.oauth2.credentials import Credentials
from google.auth.transport.requests import AuthorizedSession
from google.auth.exceptions import GoogleAuthError


class LoginController:
    @classmethod
    def validate_fields(cls, email, password):
        error_message = 'Incorrect email or password'

        user = current_app.models.User.get_user_by_email(email=email)

        if not user or not user.check_password(password):
            raise ValidationError(error_message)

        if not user.is_active:
            raise ValidationError('Your account is not active')

        return user

    @classmethod
    def login(cls, data):
        user = cls.validate_fields(**data)
        session_id = cls._create_session(user=user)
        return (session_id, user.user_id)

    @classmethod
    def login_with_social(cls, data):
```

```
        user_data = cls._authorize_user(token=data['auth_token'],
provider=data['provider'])

        user =
current_app.models.User.get_user_by_email(email=user_data['email'])

        if not user:
            user = current_app.models.User.create_user(**user_data,
is_active=True)
        elif not user.is_active:
            user.activate_user()
        session_id = cls._create_session(user=user)
        return (session_id, user.user_id)

    @classmethod
    def _authorize_user(cls, token, provider):
        if provider == 'FACEBOOK':
            return cls._authorize_with_fb(token)
        if provider == 'GOOGLE':
            return cls._authorize_with_google(token)


    @staticmethod
    def _authorize_with_fb(token):
        try:
            graph = facebook.GraphAPI(access_token=token)
            raw_data = graph.get_object(id="me", fields='first_name,
last_name, email, picture')
        except facebook.GraphAPIError as e:
            raise ValidationError(e.message)

        user_data = {
            'name': raw_data['first_name'],
            'surname': raw_data['last_name'],
            'email': raw_data['email'],
            'avatar': raw_data['picture']['data']['url']
        }
        return user_data

    @staticmethod
    def _authorize_with_google(token):
        credentials = Credentials(token)
        authed_session = AuthorizedSession(credentials)
        try:
            response =
authed_session.get('https://www.googleapis.com/oauth2/v1/userinfo?alt=json'
)
        except GoogleAuthError:
            raise ValidationError('Invalid auth token')

        raw_data = json.loads(response.text)
        user_data = {
            'name': raw_data['given_name'],
            'surname': raw_data['family_name'],
            'email': raw_data['email'],
            'avatar': raw_data['picture']
        }
        return user_data

    @classmethod
    def _create_session(cls, user):
```

```
        session_id = str(uuid.uuid1())
        login_time = 24 * 60 * 60
        started_at = time.time()
        expired_at = started_at + login_time
        session_data = {
            'user_id': user.user_id,
            'started_at': started_at,
            'expired_at': expired_at,
        }
        with redis.Redis() as redis_client:
            redis_client.set(session_id, json.dumps(session_data),
ex=login_time)

        return session_id
```

Лістинг 13. user_model.py

```
from database import db
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime

BASE_AVATAR_LINK = 'http://localhost:5000/static/images/user_avatar.png'
class User(db.Model):
    """Model for user accounts."""
    __tablename__ = 'user_profile'
    __table_args__ = {'extend_existing': True}
    user_id = db.Column(db.Integer, primary_key=True, unique=True)
    name = db.Column(db.String(15), nullable=False)
    surname = db.Column(db.String(60), nullable=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(100), nullable=True)
    avatar = db.Column(db.String(250), nullable=True)
    uuid = db.Column(db.String(36), nullable=True)
    is_active = db.Column(db.Boolean, default=False)
    registration_time = db.Column(db.DateTime, default=datetime.utcnow())

    def __repr__(self):
        return f'<User {self.name}>'

    @classmethod
    def create_user(cls, name, email, password=None, surname=None,
                    is_active=False, avatar=BASE_AVATAR_LINK):
        password_hash = generate_password_hash(password) if password else
None
        user = cls(name=name, email=email, password_hash=password_hash,
                    surname=surname, is_active=is_active, avatar=avatar)
        db.session.add(user)
        db.session.commit()
        return user

    def set_uuid(self, uuid):
        self.uuid = uuid
        db.session.add(self)
        db.session.commit()

    def delete_user(self):
        db.session.delete(self)
        db.session.commit()
```

```
@classmethod
def update_user(cls, data, user_id=user_id):
    cls.query.filter_by(user_id=user_id).update(data)
    db.session.commit()

@classmethod
def get_user_by_id(cls, user_id):
    return cls.query.filter_by(user_id=user_id).first()

@classmethod
def get_user_by_email(cls, email):
    return cls.query.filter_by(email=email).first()

@classmethod
def get_user_by_uuid(cls, uuid):
    return User.query.filter_by(uuid=uuid).first()

def set_password(self, password):
    self.password_hash = generate_password_hash(password)
    db.session.commit()

def check_password(self, password):
    print(password)
    return check_password_hash(self.password_hash, password)

def password_is_set(self):
    return self.password_hash is not None

def activate_user(self):
    self.is_active = True
    db.session.add(self)
    db.session.commit()

def change_public_fields(self, name, surname):
    self.name = name
    self.surname = surname
    db.session.add(self)
    db.session.commit()

def change_avatar_url(self, avatar):
    self.avatar = avatar
    db.session.add(self)
    db.session.commit()

def is_uuid_valid(self):
    datetime_diff = datetime.utcnow() - self.registration_time
    diff_in_hours = datetime_diff.total_seconds() / 3600
    if diff_in_hours > 24:
        return False
    return True

def get_public_data(self):
    public_data = {
        "user_id": self.user_id,
        "name": self.name,
        "surname": self.surname,
        "email": self.email,
        "avatar": self.avatar,
        "passwordIsSet": self.password_is_set(),
    }
    return public_data
```

Лістинг 14. message.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';
import { MailService } from '../_services/mail.service';

@Component({
  selector: 'app-message-list',
  templateUrl: './message-list.component.html',
  styleUrls: ['./message-list.component.css'],
})
export class MessageListComponent implements OnInit {

  panelState = false;
  recipient = new FormControl();
  subject = new FormControl();
  message = new FormControl();
  messages = {};
  types = [];
  selectedTabIndex=0;
  selected = 'Received'
  sections = [
    {value: 'Received', viewValue: 'Received'},
    {value: 'Sent', viewValue: 'Sent'},
    {value: 'Spam', viewValue: 'Spam'}
  ];
  constructor(
    private mailService: MailService,
  ) { }

  ngOnInit() {
    this.messages = {};
    this.types = [];

    this.mailService.getMessages().subscribe(m_resp => {
      this.messages = m_resp.data;
      this.mailService.getTypes().subscribe(t_resp => {
        this.types = t_resp.data;
      })
    })

  }

  createMessage(){
    let data = {
      'r_email': this.recipient.value !== null ? this.recipient.value : '',
      'subject': this.subject.value !== null ? this.subject.value : '',
      'message': this.message.value,
      'date': new Date(),
    }
    this.togglePanel();
    this.clearFields();
    this.mailService.createMessage(data).subscribe(resp => {
      alert('Message was sent');
      this.updateMessages();
    });
  }

  updateMessages(){
    this.mailService.getMessages().subscribe(m_resp => {
```

```
      this.messages = m_resp.data;
    });
  }

  clearFields(){
    this.recipient.reset();
    this.subject.reset();
    this.message.reset();
  }

  togglePanel(){
    this.panelState = !this.panelState;
  }

}
```

Лістинг 14. user-profile.component.ts

```
import {Component, OnInit, ViewChild, Injectable} from '@angular/core';
import {UserService} from "../_services/user.service";
import {Router} from "@angular/router";
import {MatSidenav} from "@angular/material/sidenav";
import {User} from '../user';
import { FormControl, Validators, FormGroup} from '@angular/forms';
import { AuthService } from '../auth/auth.service';


@Component({
  selector: 'app-user-profile',
  templateUrl: './user-profile.component.html',
  styleUrls: ['./user-profile.component.css']
})

@Injectable({
  providedIn: 'root',
})
export class UserProfileComponent implements OnInit {
  calculateFormOpened = false;
  passwordFormOpened = false;
  result: number;
  gender: string;
  userDataIsIncorrect: boolean;

  surnameFormControl = new FormControl('', [Validators.minLength(2),
Validators.maxLength(30)]);
  nameFormControl = new FormControl('', [Validators.minLength(2),
Validators.maxLength(30), Validators.required]);
  oldPassword = this.getPasswordFormControl();
  newPassword = this.getPasswordFormControl();
  confirmPassword = this.getPasswordFormControl();
  passswordGroup = new FormGroup({first: this.newPassword,
    second: this.confirmPassword}, this.passwordMatchValidator);

  @ViewChild('sidenav', {static: true}) public userSideNav: MatSidenav;
  public user;
  public editedUser = new User('','', '', '');
```

```
  private getPasswordFormControl(){
    return new FormControl('', [Validators.required,
Validators.minLength(8),
      Validators.pattern(RegExp('(?=.*[A-Za-z])(?=.*[0-9])[A-Za-z\\d]'))]);
  }

  private passwordMatchValidator(g: FormGroup) {
    return g.get('first').value === g.get('second').value
       ? null : {'mismatch': true};
 }

  updatePassword(){
    let data = {};
    if(this.oldPassword.value !== null){
      data = {'old_password': this.oldPassword.value};
    }
    data['new_password'] = this.newPassword.value
    this.userService.updatePassword(data)
    .subscribe(() => {
      this.userService.refreshUser();
      alert('Password was changed');
    });
    this.clearPasswordForm();
  }

  private clearPasswordForm(){
    this.passwordFormOpened = false;
    this.resetPasswordFields();
  }

  private resetPasswordFields(){
    this.oldPassword.reset();
    this.newPassword.reset();
    this.confirmPassword.reset();
  }

  togglePasswordForm(){
    this.passwordFormOpened = !this.passwordFormOpened;
    this.resetPasswordFields();
  }

  getPasswordErrorMessage(password){
    return password.hasError('required') ? 'You must enter a value' :
        password.hasError('minlength') ? 'Min length is 8 characters':
        password.hasError('pattern') ? 'At least 1 digit and 1 character':
''
  }

  getPasswordGroupErrorMessage(password){
    return this.passswordGroup.hasError('mismatch') ? 'Passwords do not
match.':
        this.getPasswordErrorMessage(password);
  }

  passwordDataInvalid(): boolean{
    return (this.newPassword.invalid
      || this.confirmPassword.invalid || this.passswordGroup.invalid);
  }

  logoutUser(): void {
    this.authService.userLogout()
```

```
      .subscribe(res => {
        this.authService.deleteSessionId();
        window.location.reload();
      });
  }

  private clearUser(){
    this.user = new User('','', '', '');
    this.previewUrl = null;
  }

  numberOnly(event): boolean {
      const charCode = (event.which) ? event.which : event.keyCode;
      if (charCode > 31 && (charCode < 48 || charCode > 57)) {
          return false;
      }
      return true;
  }

  letterOnly(event): boolean {
      const charCode = (event.which) ? event.which : event.keyCode;
        // console.log('charCode ', charCode);
      if ((charCode != 32) && (charCode != 39) && (charCode < 65 ||
charCode > 122 )) {
          return false;
      }
      return true;
  }

  editUser() {
    this.userService.userProfileEditable=true;
    this.previewUrl = this.user.avatar;
    this.editedUser.name = this.user.name;
    this.editedUser.surname = this.user.surname;
  }

  submitUserData()
  {
    if (this.previewUrl!=this.user.avatar){
      this.userService.updateUserAvatar(this.fileData)
          .subscribe(res => {
          });
    }
    this.userService.userProfileEditable=false;
    if(!this.editedUser.surname){this.editedUser.surname=''}
    this.userService.updateUser(this.editedUser.name,
this.editedUser.surname, this.editedUser)
        .subscribe(() => this.userService.refreshUser());
  }


  checkUserData()
  {
    let nameIsIncorect = this.nameFormControl.hasError('minlength')||
        this.nameFormControl.hasError('maxlength')||
        this.nameFormControl.hasError('required');
    let surnameIsCorrect = this.surnameFormControl.hasError('minlength')||
        this.surnameFormControl.hasError('maxlength');
    this.userDataIsIncorrect = nameIsIncorect||surnameIsCorrect;
  }
```

```
  CancelEdit(){
    this.userService.userProfileEditable=false;
    this.editedUser.name = this.user.name;
    this.editedUser.surname = this.user.surname;
    this.previewUrl = this.user.avatar;
  }

  fileData: File = null;
  previewUrl:any = null;
  fileProgress(fileInput: any) {
    this.fileData = <File>fileInput.target.files[0];
    this.preview();
  }

  preview() {
    // Show preview
    var mimeType = this.fileData.type;
    if (mimeType.match(/image\/*/) == null) {
      return;
    }

    var reader = new FileReader();
    reader.readAsDataURL(this.fileData);
    reader.onload = (_event) => {
      this.previewUrl = reader.result;
      console.log(this.previewUrl);
    }
  }

  constructor(
      private userService: UserService,
      private router: Router,
      private authService: AuthService,
  ){ }

  ngOnInit() {
    this.clearUser();
    this.userService.refreshUser();
    this.userService.setUserSideNav(this.userSideNav);
    this.userService.getEmittedValue()
        .subscribe(item => this.user=item);
  }

}
```

Лістинг 15. mail.service.ts

```
import { Injectable } from '@angular/core';
import { HttpHeaders, HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthService } from '../auth/auth.service';
import { BASE_URL } from './config';
import { ErrorService } from './error.service';
import { catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
```

```
})
export class MailService {

  private messageUrl = BASE_URL + '/mail/v1/messages';  // URL to web api
  private typesUrl = BASE_URL + '/mail/v1/types';  // URL to web api

  constructor(
    private authService: AuthService,
    private http: HttpClient,
    private errorService: ErrorService,
  ) { }

createMessage(data): Observable<any> {
  let header = new HttpHeaders({'Authorization':
this.authService.getSessionId()});
  return this.http.post(this.messageUrl, data, {headers: header})
  .pipe(
    catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
  );
}

getMessages(): Observable<any> {
  let header = new HttpHeaders({'Authorization':
this.authService.getSessionId()});
  return this.http.get(this.messageUrl, {headers: header})
  .pipe(
    catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
  );
}

getTypes(): Observable<any> {
  let header = new HttpHeaders({'Authorization':
this.authService.getSessionId()});
  return this.http.get(this.typesUrl, {headers: header})
  .pipe(
    catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
  );
}

}
```

Лістинг 16. user.service.ts

```
import { Injectable, EventEmitter, Output} from '@angular/core';
import {HttpClient, HttpHeaders} from '@angular/common/http'
import { Observable } from 'rxjs';
import {MatSidenav} from "@angular/material/sidenav";
import { BASE_URL } from './config'
import { catchError } from 'rxjs/operators';
import { AuthService } from '../auth/auth.service';
import { ErrorService } from './error.service';


@Injectable({
```

```
  providedIn: 'root'
})
export class UserService {

  private userUrl = BASE_URL + '/user/v1/user';
  private confirmationUrl = BASE_URL + '/otc/v1/otc/';
  private userAvatarUrl = `${this.userUrl}/avatar`;

  @Output() userDataEmitter: EventEmitter<any> = new EventEmitter();

  userSideNav: MatSidenav;
  userProfileEditable = false;

  setUserProfile(user){
    this.userDataEmitter.emit(user.data);
  }

  refreshUser(){
    if(this.authService.userIsAuthorized()){
      this.getUserProfile().subscribe(resp => {
        this.setUserProfile(resp.body);
      });
    }
  }

  getEmittedValue(){
    return this.userDataEmitter;
  }

  uuidConfirmation(uuid) {
    return this.http.patch(this.confirmationUrl + uuid, null);
  }

  postCredentials(data): Observable<any> {
    return this.http.post(this.userUrl, data)
    .pipe(
      catchError(this.errorService.handleError)
    );
  }

  getUserProfile() {
    let header = new HttpHeaders({'Authorization':
localStorage.getItem('sessionId')});
    return this.http.get(this.userUrl, {headers: header, observe:
'response'})
    .pipe(
      catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
    );
  }

  updatePassword(data): Observable<any> {
    let header = new HttpHeaders({'Authorization':
localStorage.getItem('sessionId')});
    return this.http.patch(this.userUrl, data, {headers: header, observe:
'response'})
    .pipe(
      catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
    );
  }
```

```
  updateUser(name, surname, capacity): Observable<any> {
      let header = new HttpHeaders({'Authorization':
localStorage.getItem('sessionId')});
      const url = this.userUrl;
      return this.http.put(url, {name, surname, capacity}, {headers:
header, observe: 'response'})
      .pipe(
          catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
      );
  }

  updateUserAvatar(avatar): Observable<any> {
    let header = new HttpHeaders({'Authorization':
localStorage.getItem('sessionId')});
    const url = this.userAvatarUrl;
    const formData = new FormData();
    formData.append('file', avatar);
    return this.http.post(url, formData, {headers: header, observe:
'response'})
        .pipe(
            catchError((err) => this.errorService.handleError(err,
this.authService.getSessionId()))
        );
  }

  setUserSideNav(sideNav: MatSidenav){
    this.userSideNav = sideNav;
  }

  toggleUserProfile(){
    this.userSideNav.toggle();
    this.userProfileEditable = false;
  }

  closeUserProfile(){
    this.userSideNav.close()
  }

  getUserId(): number {
    return Number(localStorage.getItem('userId'))
  }

  constructor(
    private http: HttpClient,
    private authService: AuthService,
    private errorService: ErrorService,
  ) { }
}
```

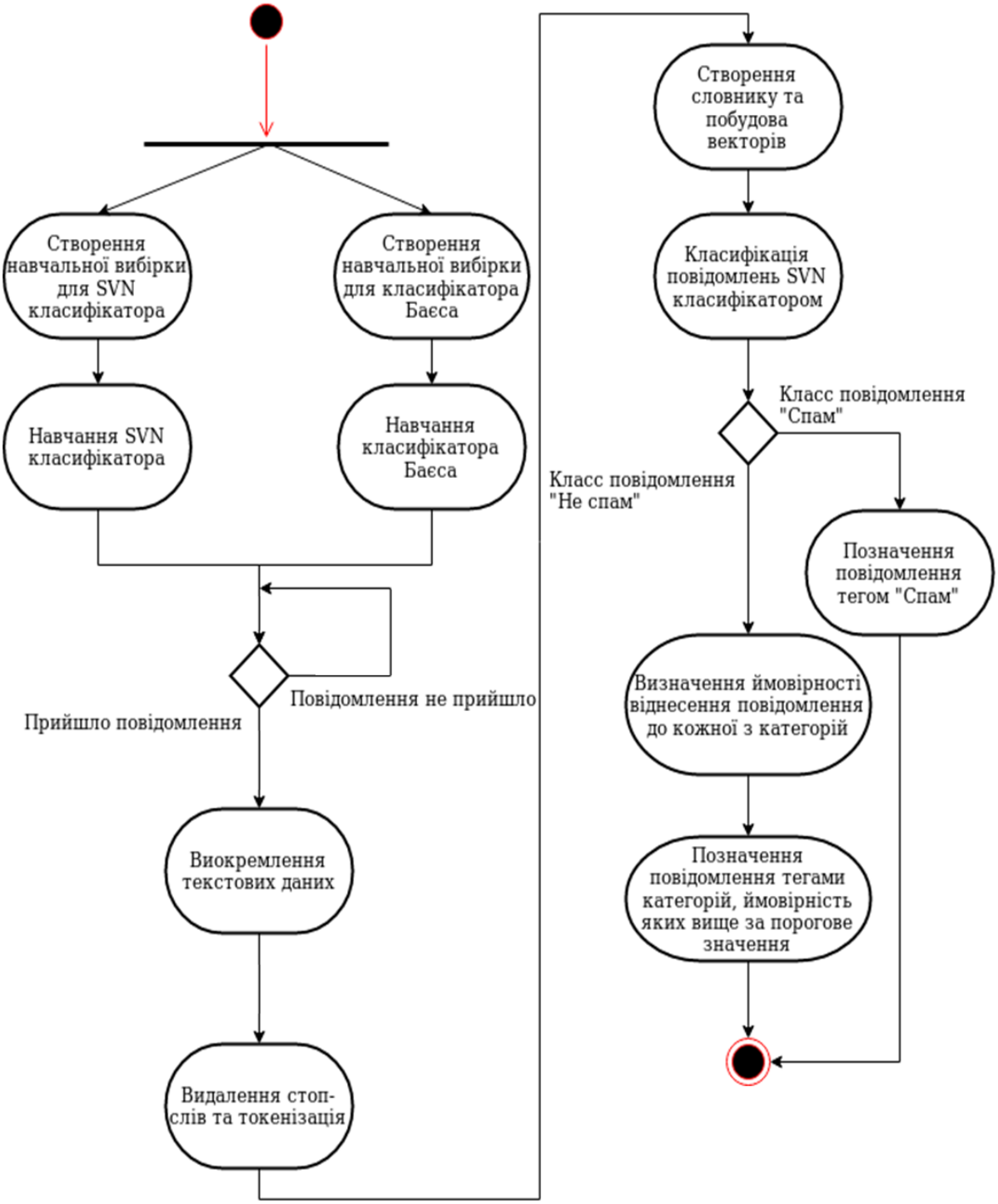
Лістинг 16. auth.service.ts

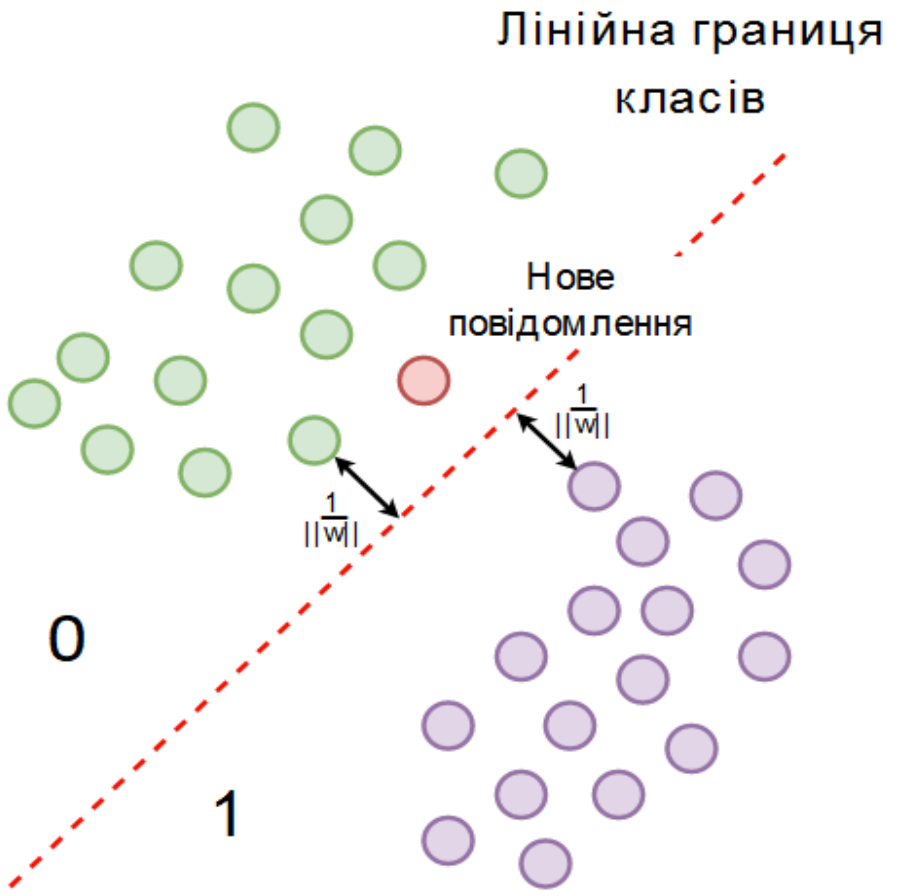
```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { HttpHeaders, HttpClient } from '@angular/common/http';
import { BASE_URL } from '../_services/config'
```

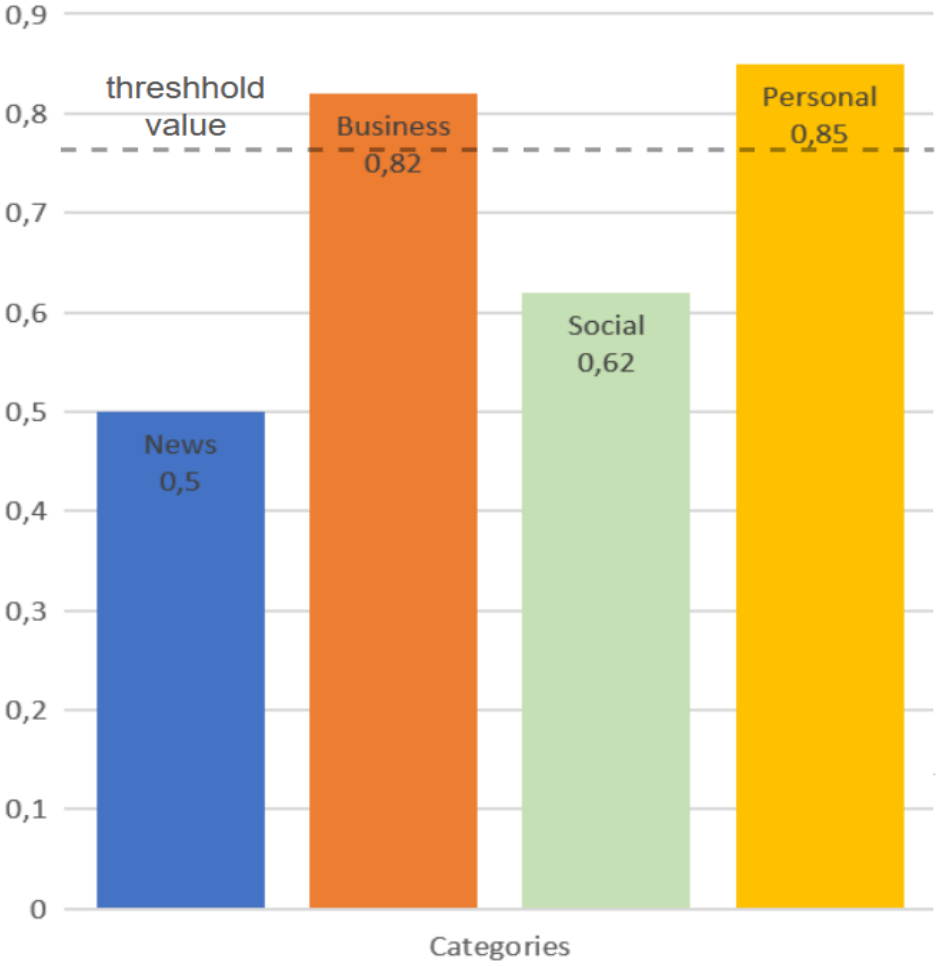
```
import { catchError } from 'rxjs/operators';
import { ErrorService } from '../_services/error.service';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private loginUrl = BASE_URL + '/user/v1/login';
  private logoutUrl = BASE_URL + '/user/v1/logout';
  private sessionTimeout;
  private day = 1000 * 60 * 60 * 24;

  userLogin(data): Observable<any> {
    return this.http.post(this.loginUrl, data, {observe: 'response'})
    .pipe(
      catchError(this.errorService.handleError.bind(this))
    );
  }

  userLogout(): Observable<any> {
    let header = new HttpHeaders({'Authorization':
localStorage.getItem('sessionId')});
    console.log(header);
    return this.http.post(this.logoutUrl, null, {headers: header, observe:
'response'})
    .pipe(
      catchError(this.errorService.handleError.bind(this))
    );
  }

  private clearSession(){
    this.deleteSessionId();
    // window.location.reload();
  }

  setSessionId(sessionId, userId) {
    localStorage.setItem('sessionId', sessionId);
    localStorage.setItem('userId', userId);
    // Set timeout to clear sessionId after 24 hours
    this.sessionTimeout = setTimeout(this.clearSession.bind(this),
this.day);
  }

  userIsAuthorized(): boolean {
    return localStorage.getItem('sessionId') !== null;
  }

  deleteSessionId() {
    clearTimeout(this.sessionTimeout);
    return localStorage.removeItem('sessionId');
  }

  getSessionId() {
    return localStorage.getItem('sessionId');
  }

  constructor(
    private http: HttpClient,
    private errorService: ErrorService,
    ) { }
}
```

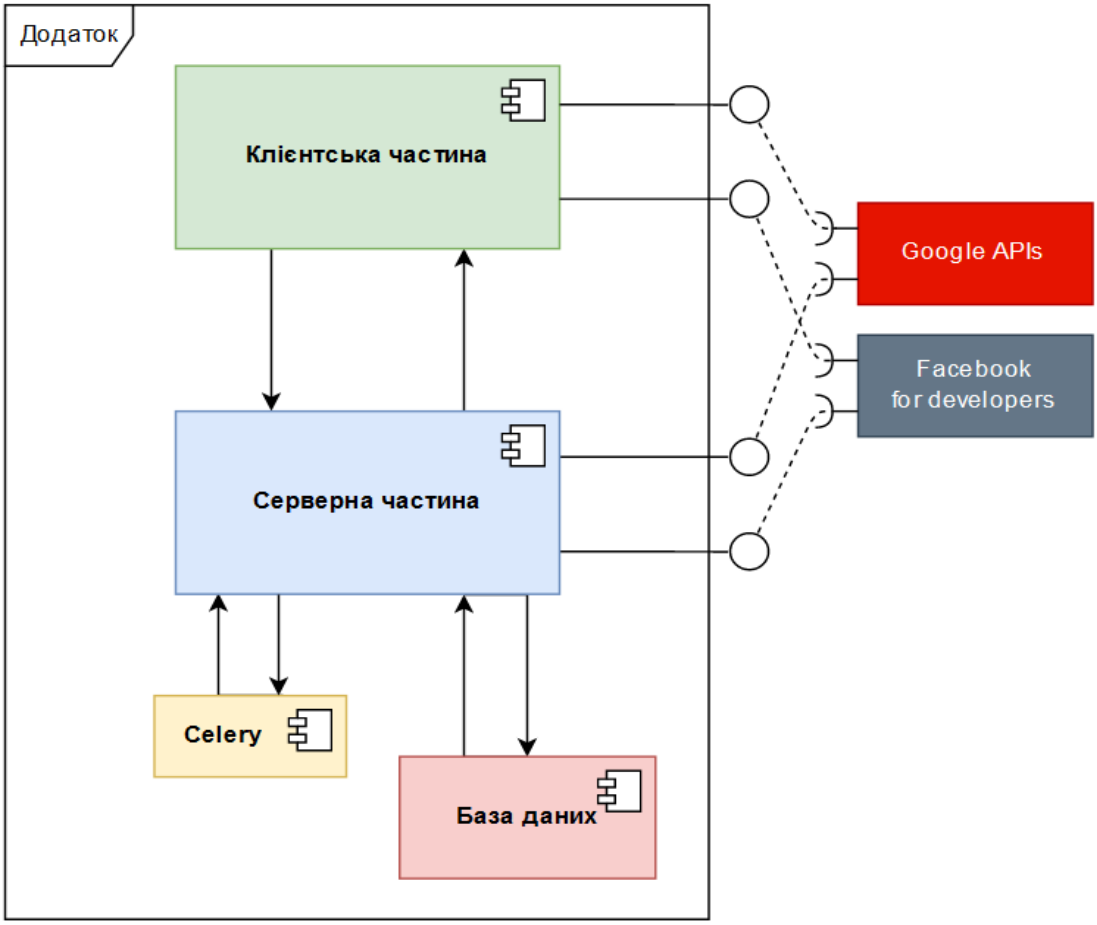
Додаток 2

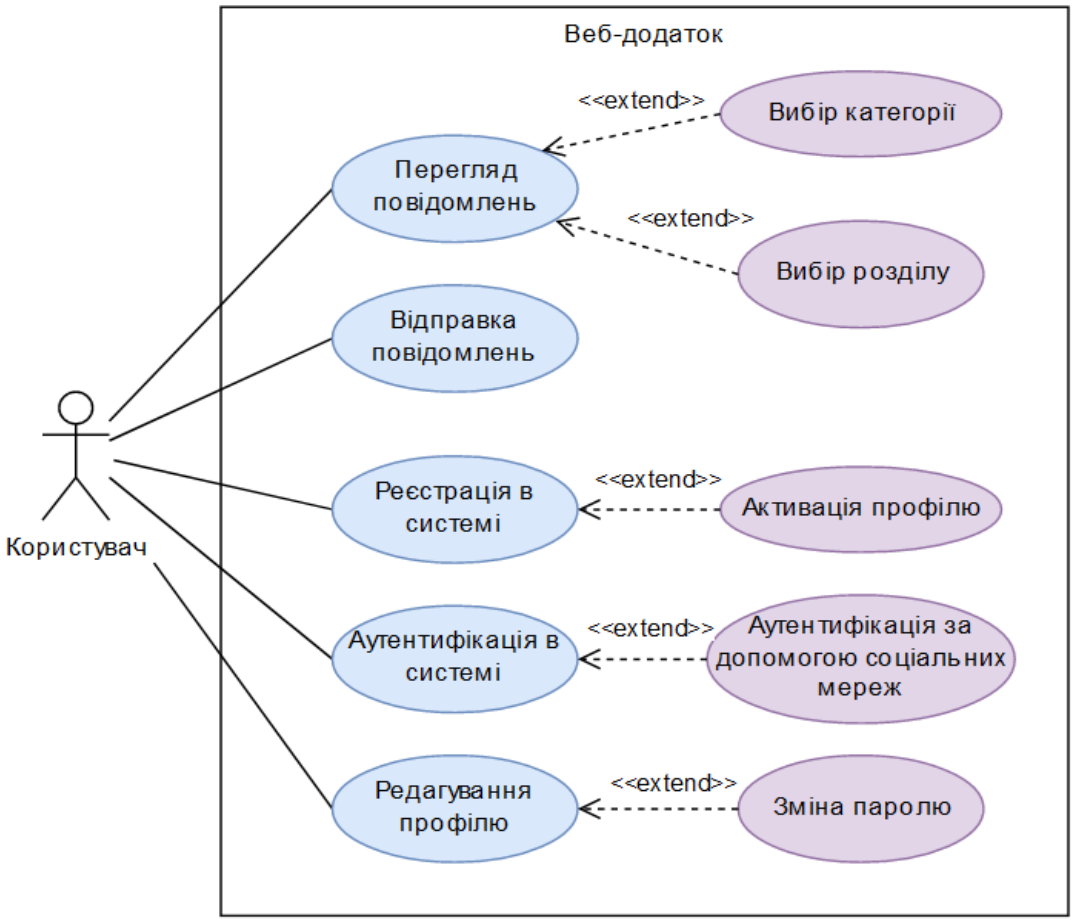
Копії графічних матеріалів

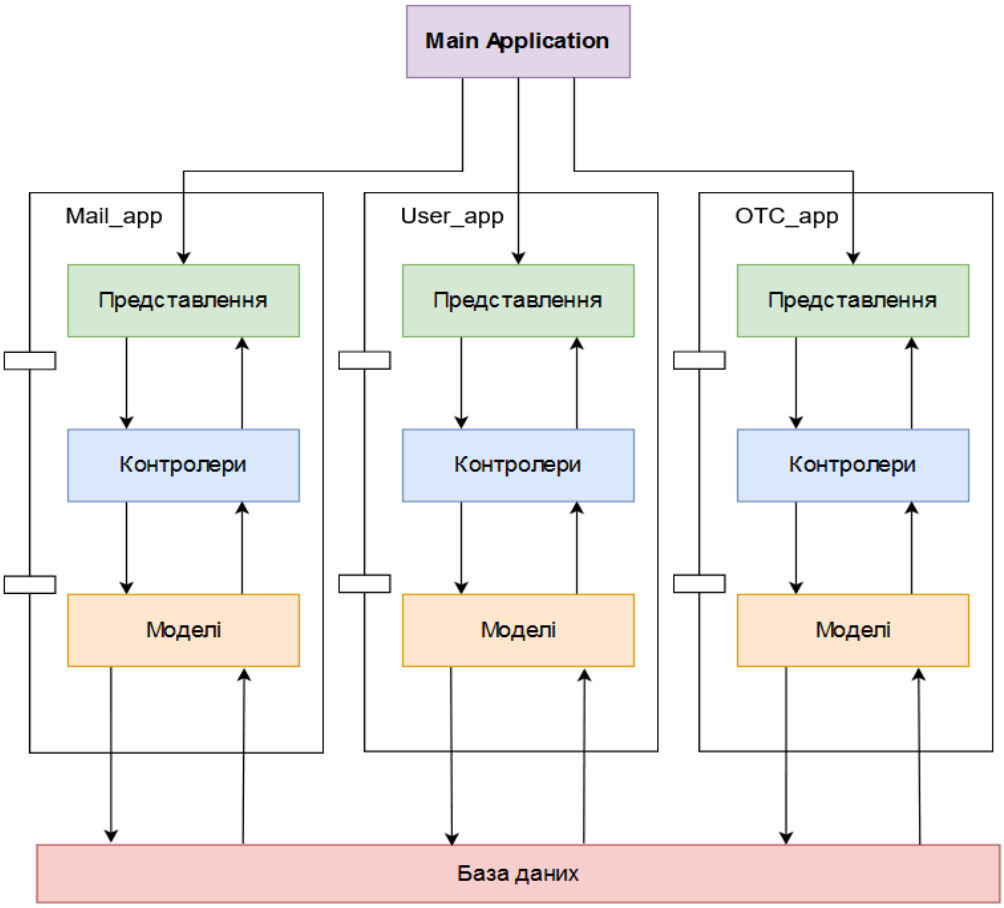

Створення навчальної вибірки для SVN класифікатора
Створення навчальної вибірки для класифікатора Баєса
Навчання SVN класифікатора
Навчання класифікатора Баєса
Повідомлення не прийшло
Прийшло повідомлення
Виокремлення текстових даних
Видалення стоп-слів та токенізація
Створення словнику та побудова векторів
Класифікація повідомлень SVN класифікатором
Класс повідомлення "Спам"
Класс повідомлення "Не спам"
Позначення повідомлення тегом "Спам"
Визначення ймовірності віднесення повідомлення до кожної з категорій
Позначення повідомлення тегами категорій, ймовірність яких вище за порогове значення

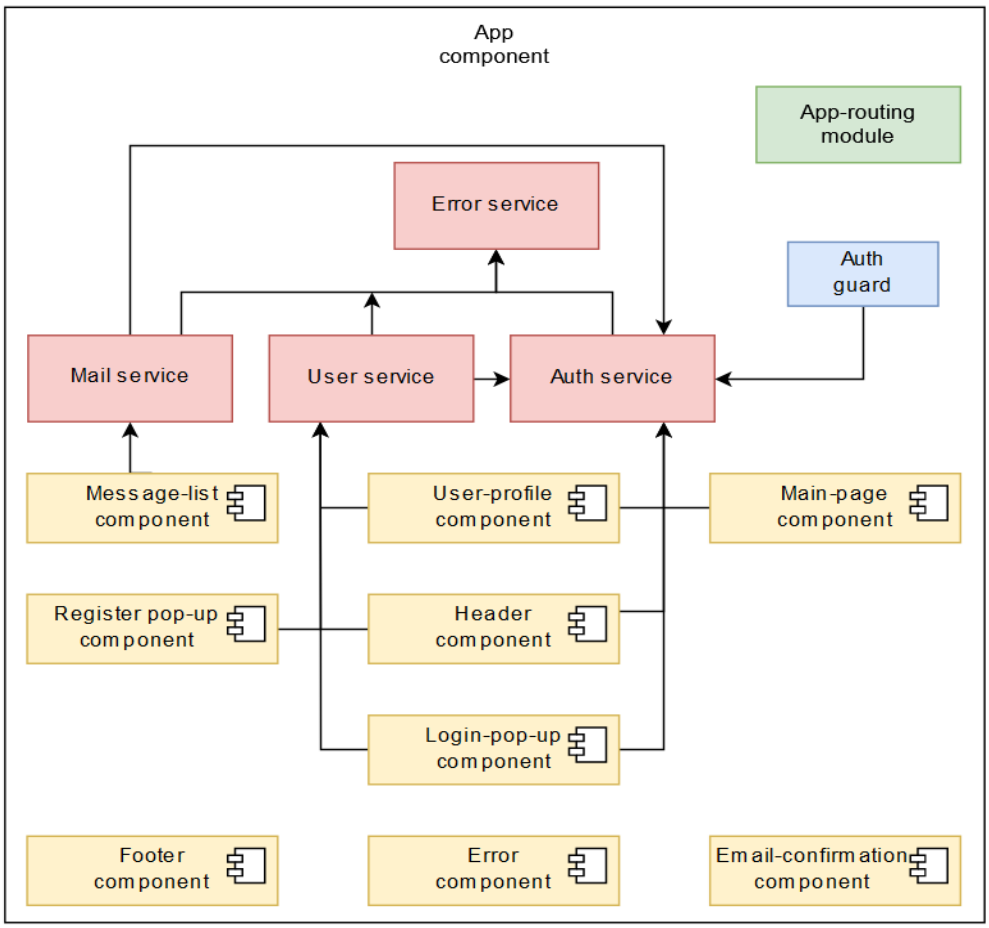

Лінійна границя класів
Нове повідомлення
1/||w||
1/||w||
0
1

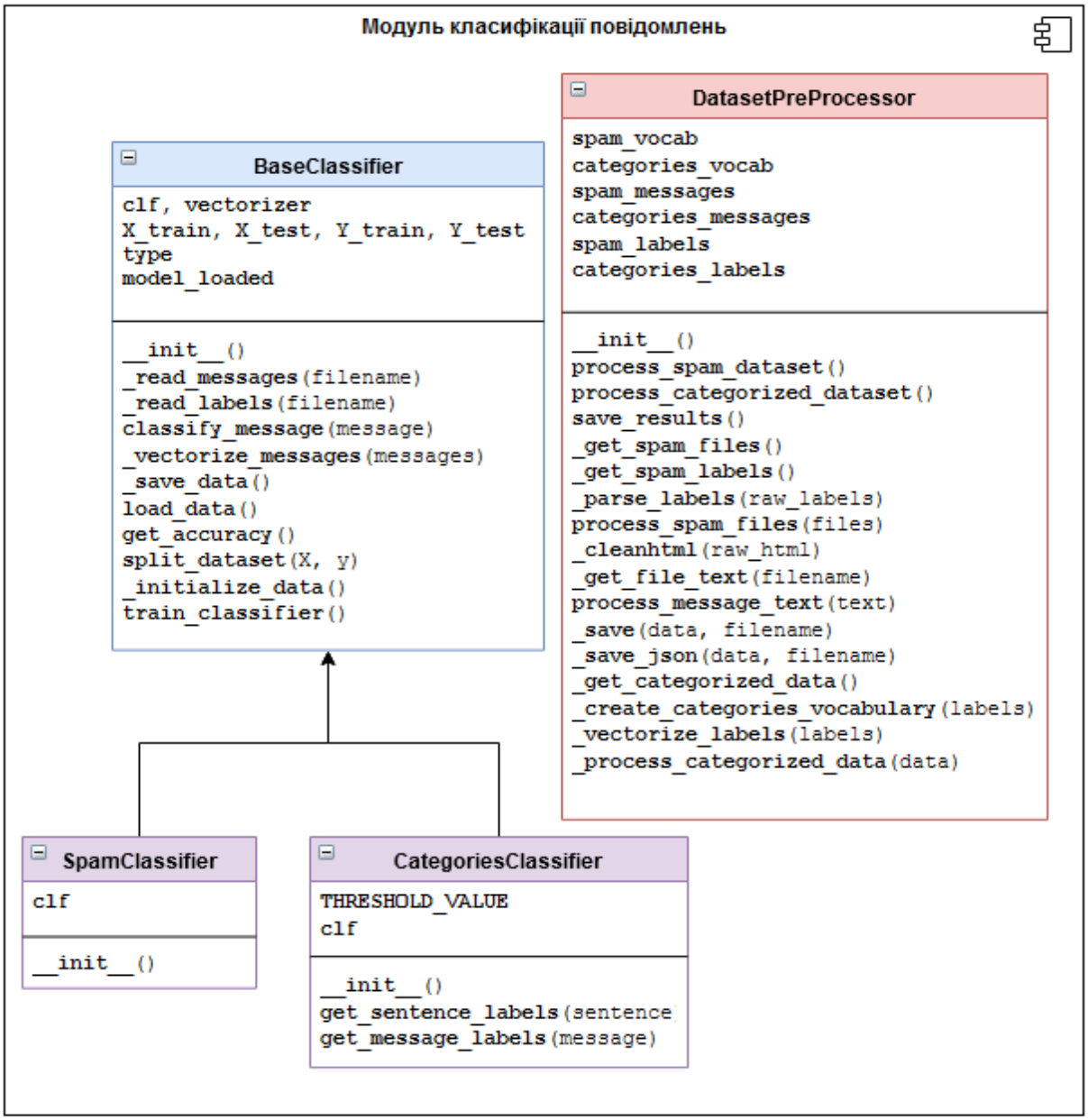

0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
threshhold value
News 0,5
Business 0,82
Social 0,62
Personal 0,85
Categories

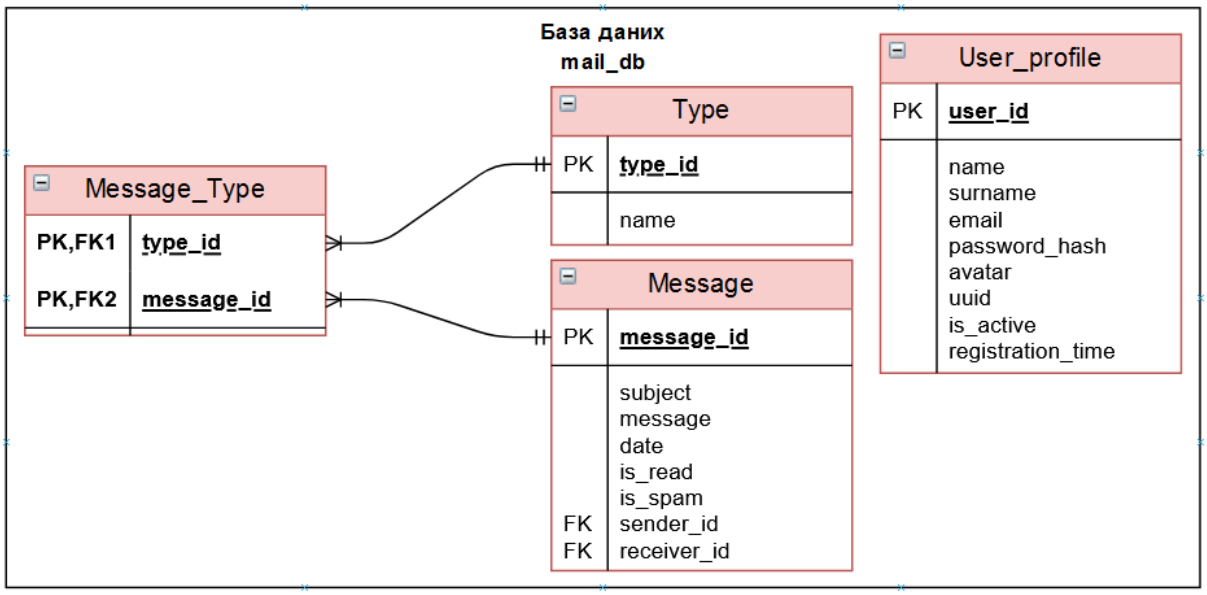

Додаток
Клієнтська частина
Серверна частина
Celery
База даних
Google APIs
Facebook
for developers

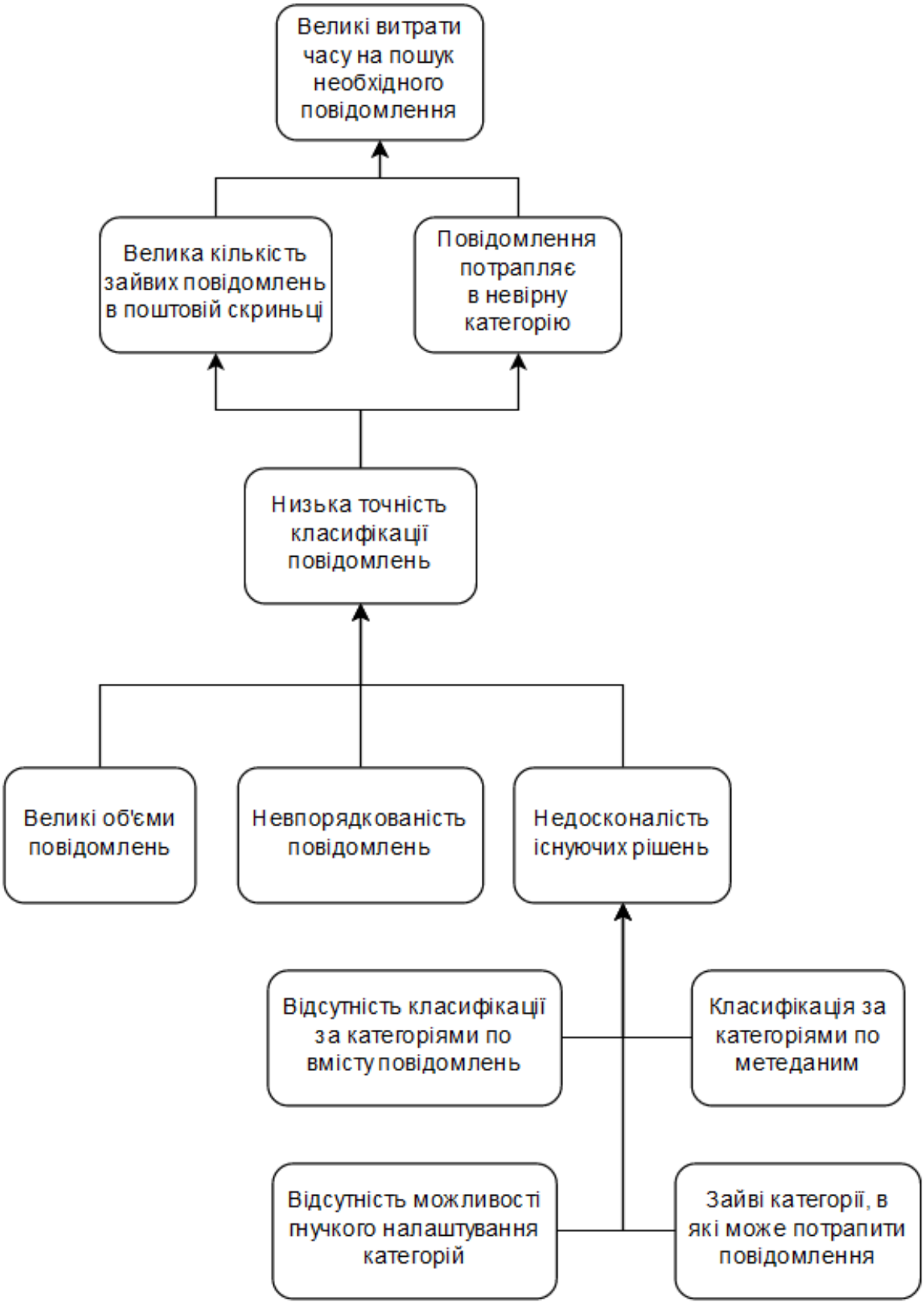

Веб-додаток
Користувач
Перегляд повідомлень
Відправка повідомлень
Реєстрація в системі
Аутентифікація в системі
Редагування профілю
Вибір категорії
Вибір розділу
Активація профілю
Аутентифікація за допомогою соціальних мереж
Зміна паролю
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>


Main Application
Mail_app
User_app
OTC_app
Представлення
Представлення
Представлення
Контролери
Контролери
Контролери
Моделі
Моделі
Моделі
База даних


App component
App-routing module
Error service
Auth guard
Mail service
User service
Auth service
Message-list component
User-profile component
Main-page component
Register pop-up component
Header component
Login-pop-up component
Footer component
Error component
Email-confirmation component


Модуль класифікації повідомлень
BaseClassifier
clf, vectorizer
X_train, X_test, Y_train, Y_test
type
model_loaded
__init__()
_read_messages(filename)
_read_labels(filename)
classify_message(message)
_vectorize_messages(messages)
_save_data()
load_data()
get_accuracy()
split_dataset(X, y)
_initialize_data()
train_classifier()
DatasetPreProcessor
spam_vocab
categories_vocab
spam_messages
categories_messages
spam_labels
categories_labels
__init__()
process_spam_dataset()
process_categorized_dataset()
save_results()
_get_spam_files()
_get_spam_labels()
_parse_labels(raw_labels)
process_spam_files(files)
_cleanhtml(raw_html)
_get_file_text(filename)
process_message_text(text)
_save(data, filename)
_save_json(data, filename)
_get_categorized_data()
_create_categories_vocabulary(labels)
_vectorize_labels(labels)
_process_categorized_data(data)
SpamClassifier
clf
__init__()
CategoriesClassifier
THRESHOLD_VALUE
clf
__init__()
get_sentence_labels(sentence
get_message_labels(message)


База даних
mail_db
Message_Type
PK,FK1 type_id
PK,FK2 message_id
Type
PK type_id
name
Message
PK message_id
subject
message
date
is_read
is_spam
FK sender_id
FK receiver_id
User_profile
PK user_id
name
surname
email
password_hash
avatar
uuid
is_active
registration_time


Великі витрати часу на пошук необхідного повідомлення
Велика кількість зайвих повідомлень в поштовій скриньці
Повідомлення потрапляє в невірну категорію
Низька точність класифікації повідомлень
Великі об'єми повідомлень
Невпорядкованість повідомлень
Недосконалість існуючих рішень
Відсутність класифікації за категоріями по вмісту повідомлень
Класифікація за категоріями по метеданим
Відсутність можливості гнучкого налаштування категорій
Зайві категорії, в які може потрапити повідомлення

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

КОМБІНОВАНИЙ МЕТОД АВТОМАТИЗОВАНОЇ КЛАСИФІКАЦІЇ ПОВІДОМЛЕНЬ ЕЛЕКТРОННОЇ ПОШТИ

Виконав: Редін Кирило Артурович

Науковий керівник: доцент, к.т.н. Заболотня Тетяна Миколаївна

Київ – 2019

АКТУАЛЬНІСТЬ

- Збільшення кількості користувачів електронних поштових систем;
- Великий відсоток спам повідомлень (понад 20% на 2019 рік);
- Зростання середньої кількості повідомлень електронної пошти, які користувач отримує кожного дня;
- Недостатня точність існуючих методів для класифікації повідомлень електронної пошти.

Об’єкт та предмет дослідження

- **Об’єктом дослідження** є процес класифікації природномовних текстових даних електронної пошти.

- **Предметом дослідження** є методи автоматизованої класифікації текстових даних.

Наукове завдання: розробити метод класифікації текстового вмісту повідомлень електронної пошти.

Мета дослідження: підвищення точності автоматизованої класифікації текстового вмісту повідомлень електронної пошти.

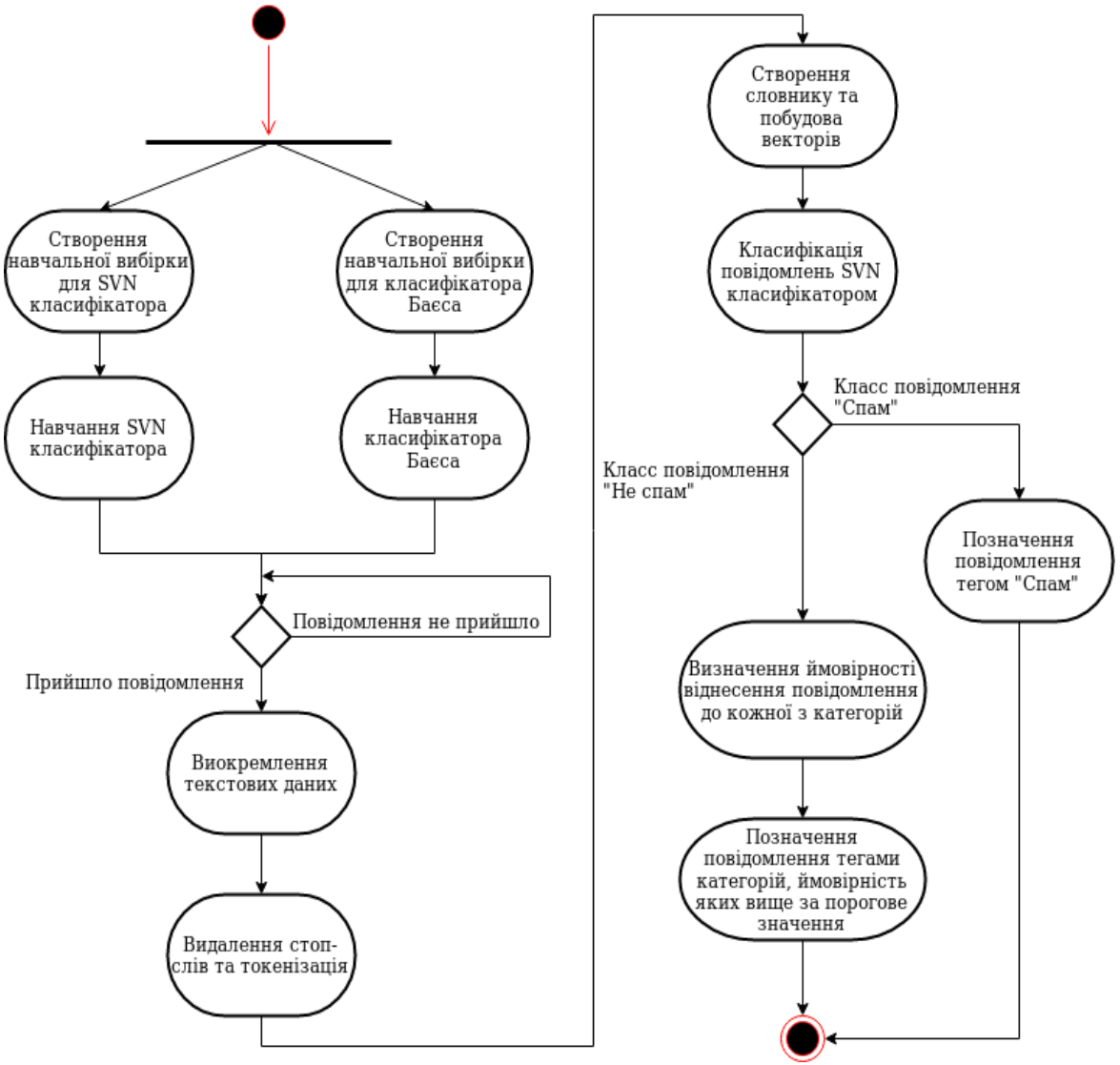
Окремі завдання

1. Проаналізувати існуючі методи автоматизованої класифікації тексту.
2. На основі проведеного аналізу розробити власний комбінований метод класифікації текстового вмісту повідомлень електронної пошти.
3. Реалізувати запропонований метод у вигляді програмного забезпечення.
4. Провести аналіз результатів реалізованого методу.
5. Побудувати бізнес-модель для представлення структурних, операційних та фінансових механізмів роботи для створення програмного продукту.

АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ КЛАСИФІКАЦІЇ ПОВІДОМЛЕНЬ

	Фільтрація спаму	Класифікація за категоріями	Виконання в реальному часі
Метод Баєса	3	5	5
Метод опорних векторів	5	2	5
Метод k-найближчих сусідів	3	5	2
Дерева рішень	3	3	4
Випадковий ліс	3	4	4
Одношаровий перцептрон	3	3	5
Багатошаровий перцептрон	4	4	5
Мережа радіальних базисних функцій	4	4	5

СТРУКТУРА КОМБІНОВАНОГО МЕТОДУ КЛАСИФІКАЦІЇ ПОВІДОМЛЕНЬ



ЕТАП ОЧИЩЕННЯ ТЕКСТОВИХ ДАНИХ

Hi there!

Remember that you are waiting for tomorrow's **Machine Learning Meet-up!**
Day and time: August 14, 19: 30-21: 00

Place: Ring Ukraine, st. Antonovich 81 , Tower C, 8th floor.

Contacts for operational communications:

Anna 096 707 78 80 Natalia 095 272 27 33

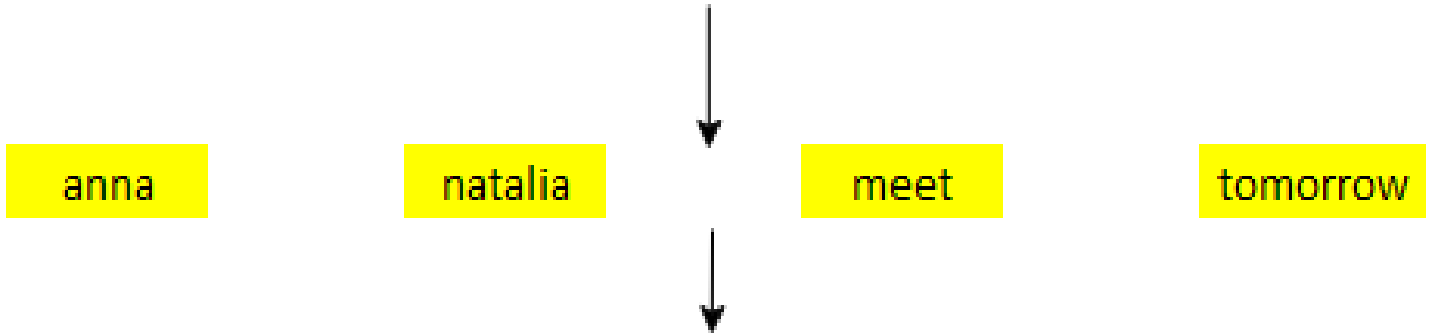
Please inform us if your plans have changed on August 14.

See you on meeting!

hi there remember you wait tomorrow machine learn meet up day time august 14 19:30 21:00 place ring ukraine street antonovich 81 tower 8th floor contact operation communication anna natalia please inform us you plan change august 14 see you meet

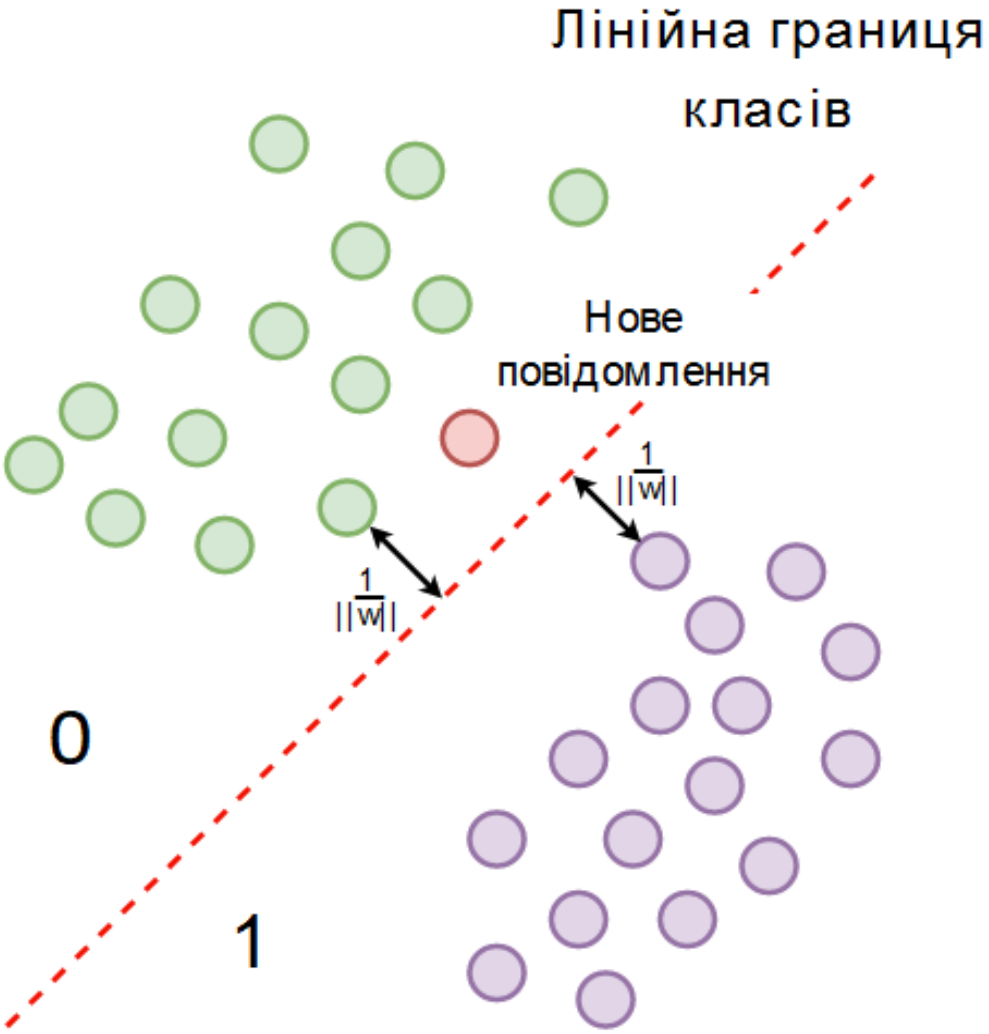
ВЕКТОРИЗАЦІЯ ПОВІДОМЛЕННЯ

Anna and natalia have a meeting tomorrow.



anna	natalia	meet	start	day	place	tomorrow
1	1	1	0	0	0	1

ЕТАП ФІЛЬТРАЦІЇ ВІД СПАМУ



ЕТАП КЛАСИФІКАЦІЇ ЗА КАТЕГОРІЯМИ

$$P(c|d) = P(d|c)P(c)$$

$$P(d|c) = \prod_{i=1}^{n} P(w_i|c)$$

$$P(w_i|c) = \frac{W_{ic} + 1}{\sum_{k \in N} W_{kc} + N}$$

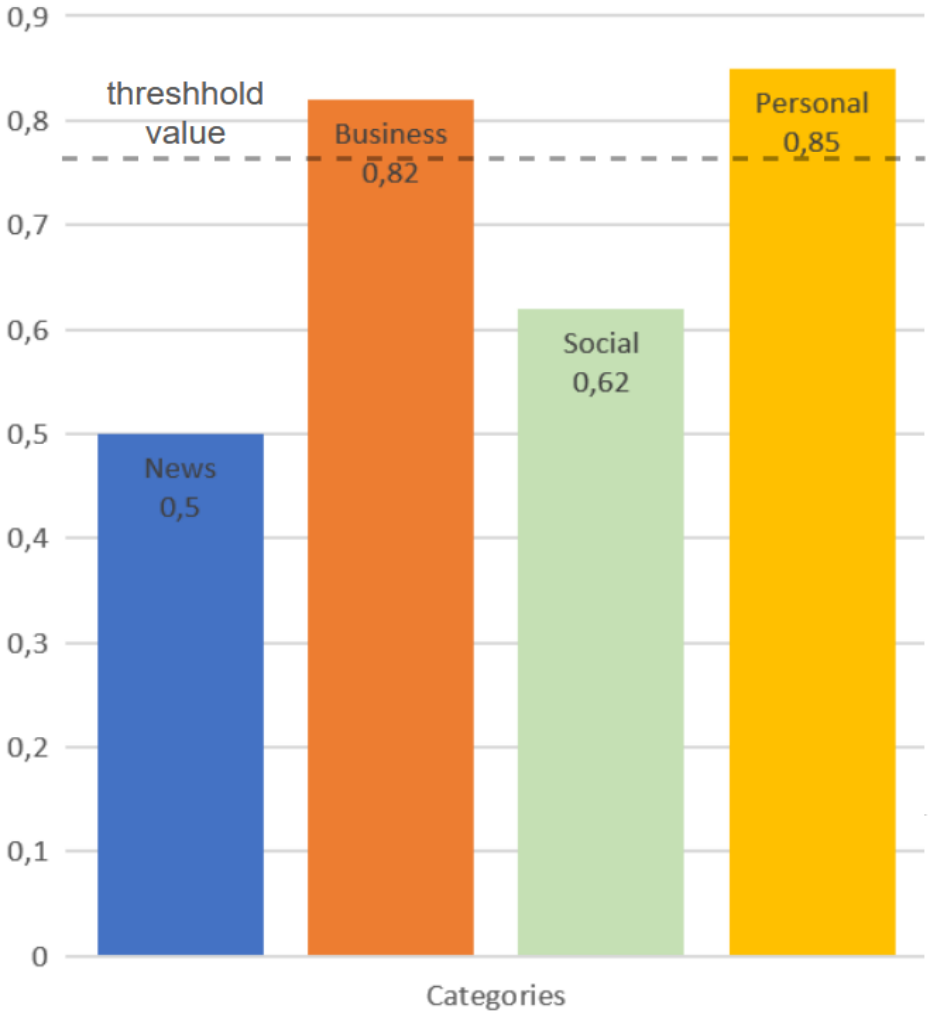
$$q_c = \ln\big(P(d|c)P(c)\big)$$

$$P(c) = \frac{M_c}{M}$$

$$q_c = \sum_{i=1}^{n} \ln\left(\frac{W_{ic} + 1}{\sum_{k \in N} W_{kc} + N}\right) + \ln\left(\frac{M_c}{M}\right)$$

$$P(c|d) = \frac{e^{q_c}}{\sum_{c_k \in C} e^{q_{c_k}}}$$

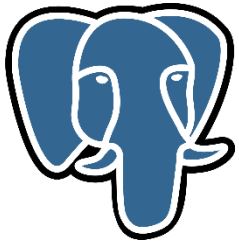
ГРАНИЧНЕ ЗНАЧЕННЯ ЙМОВІРНОСТІ



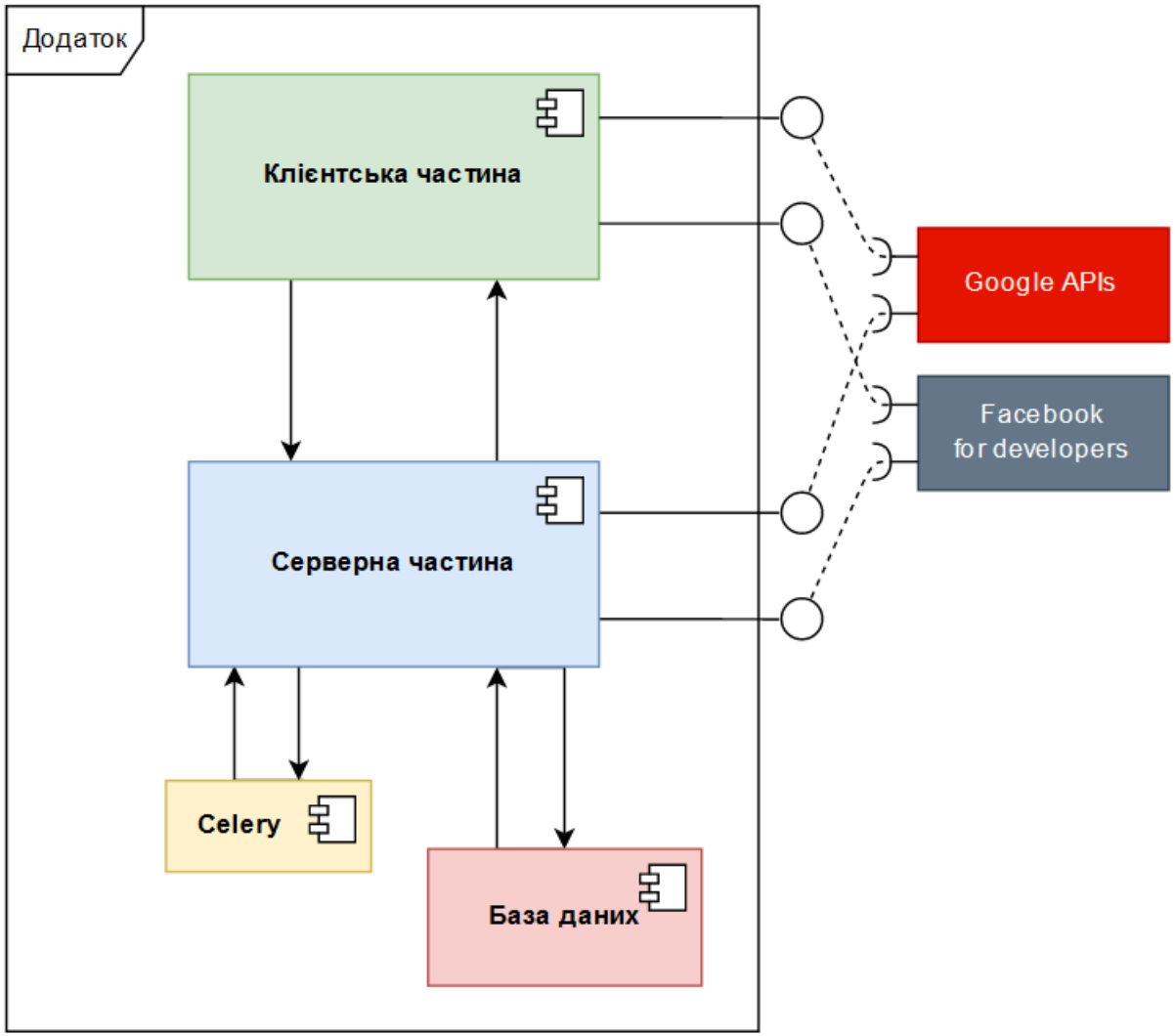
ОБРАНІ ТЕХНОЛОГІЇ

- Серверна частина:
  - Python
  - Flask
  - NLTK
  - Scikit-learn
  - PostgreSQL
- Клієнтська частина
  - TypeScript
  - Angular 8
  - AngularMaterial UI

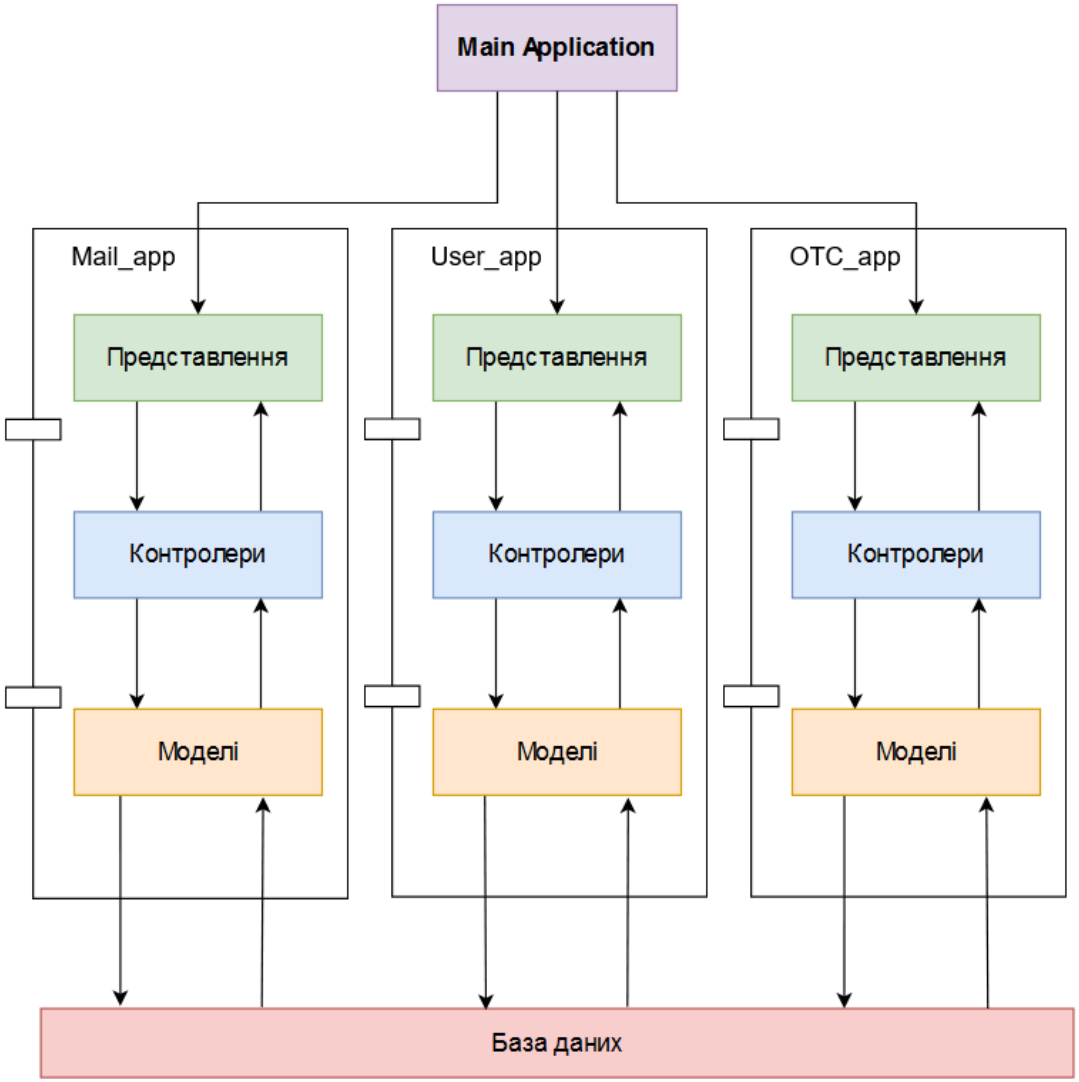




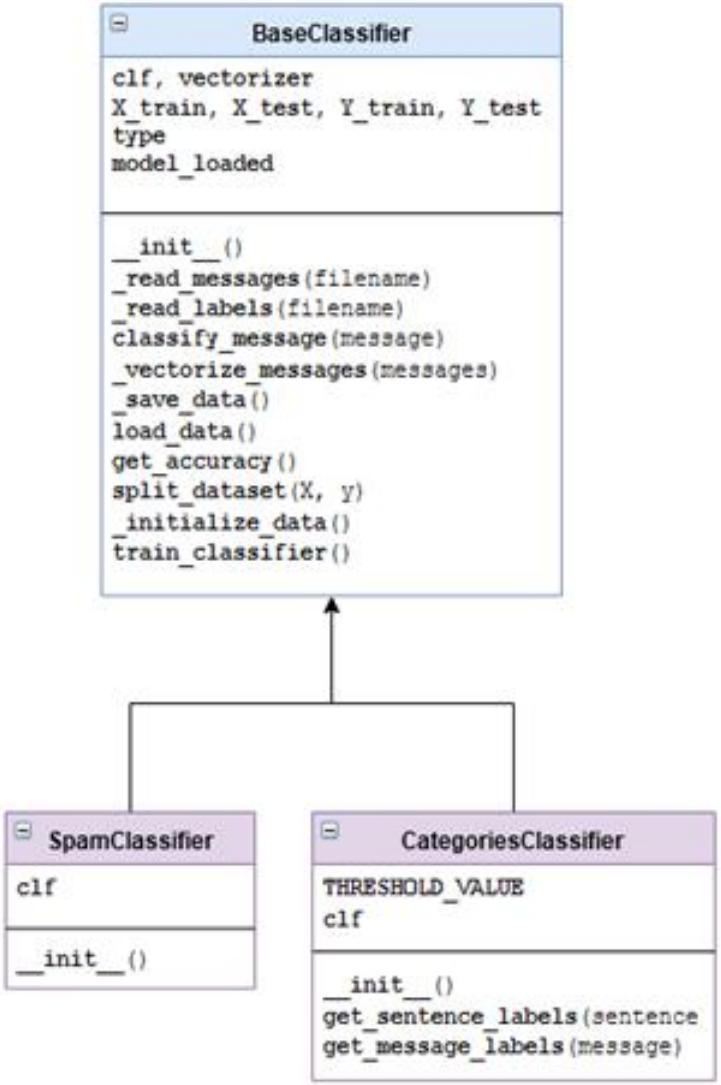

АРХІТЕКТУРА РОЗРОБЛЕНОЇ СИСТЕМИ

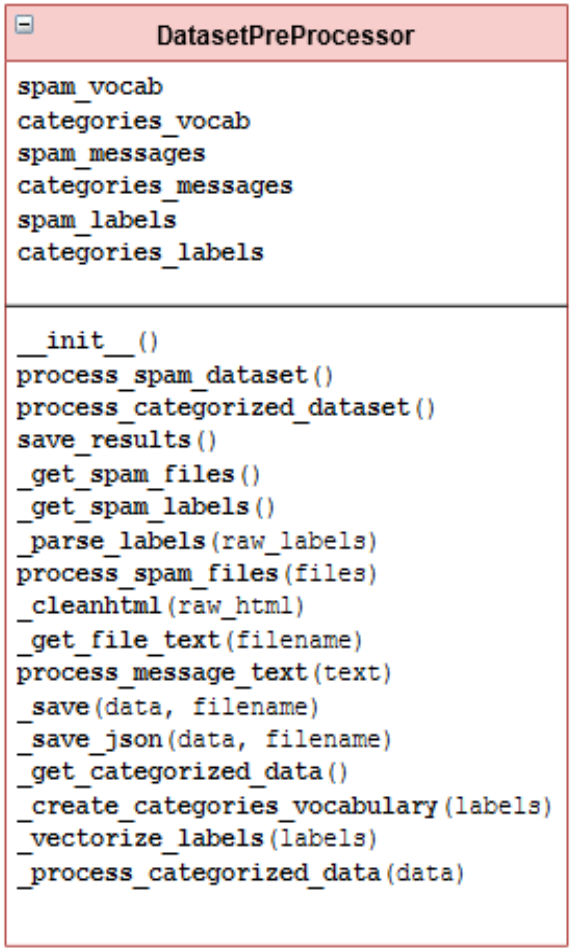


СТРУКТУРА СЕРВЕРНОЇ ЧАСТИНИ

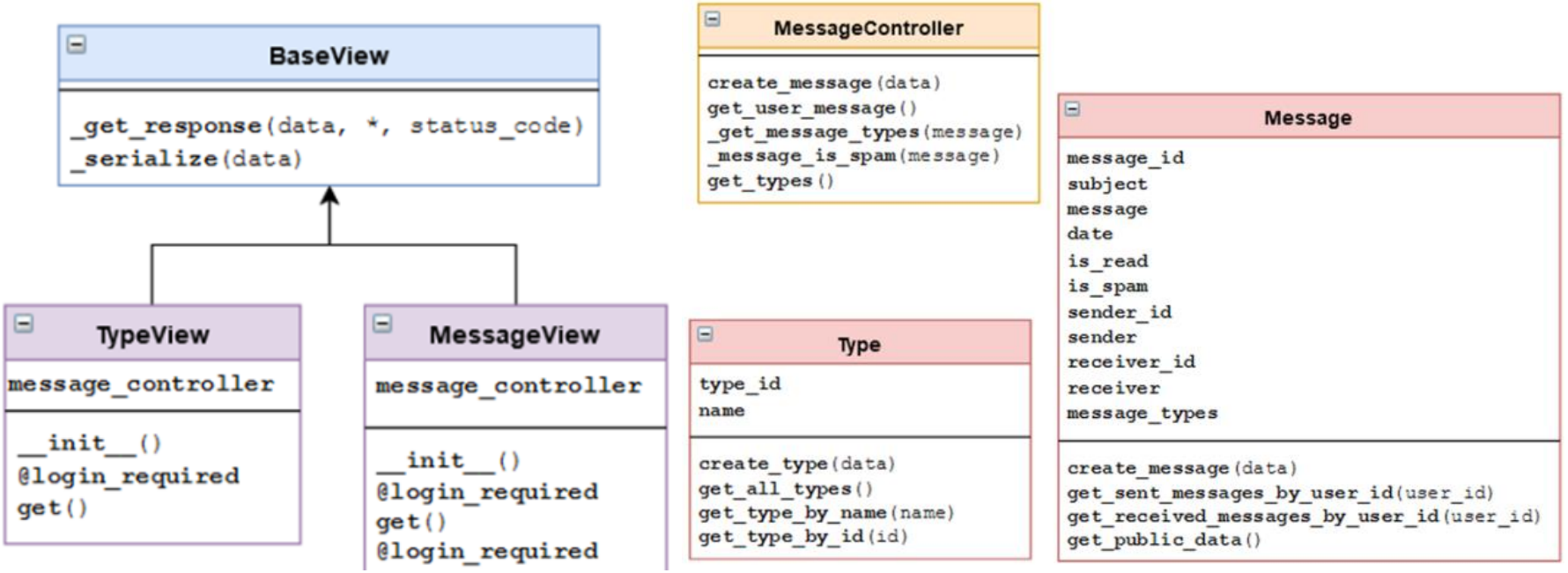


МОДУЛЬ КЛАСИФІКАЦІЇ ПОВІДОМЛЕНЬ

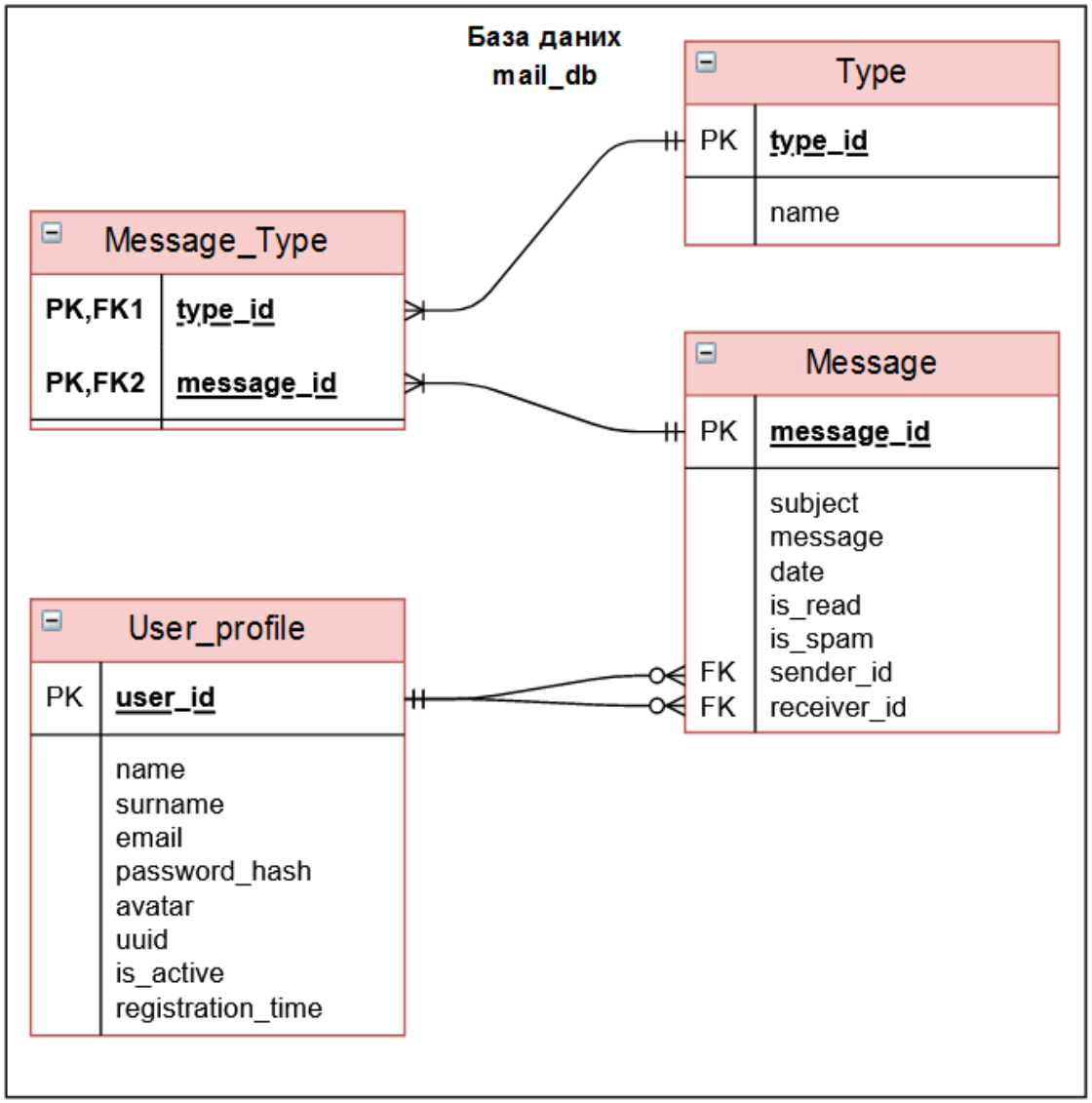




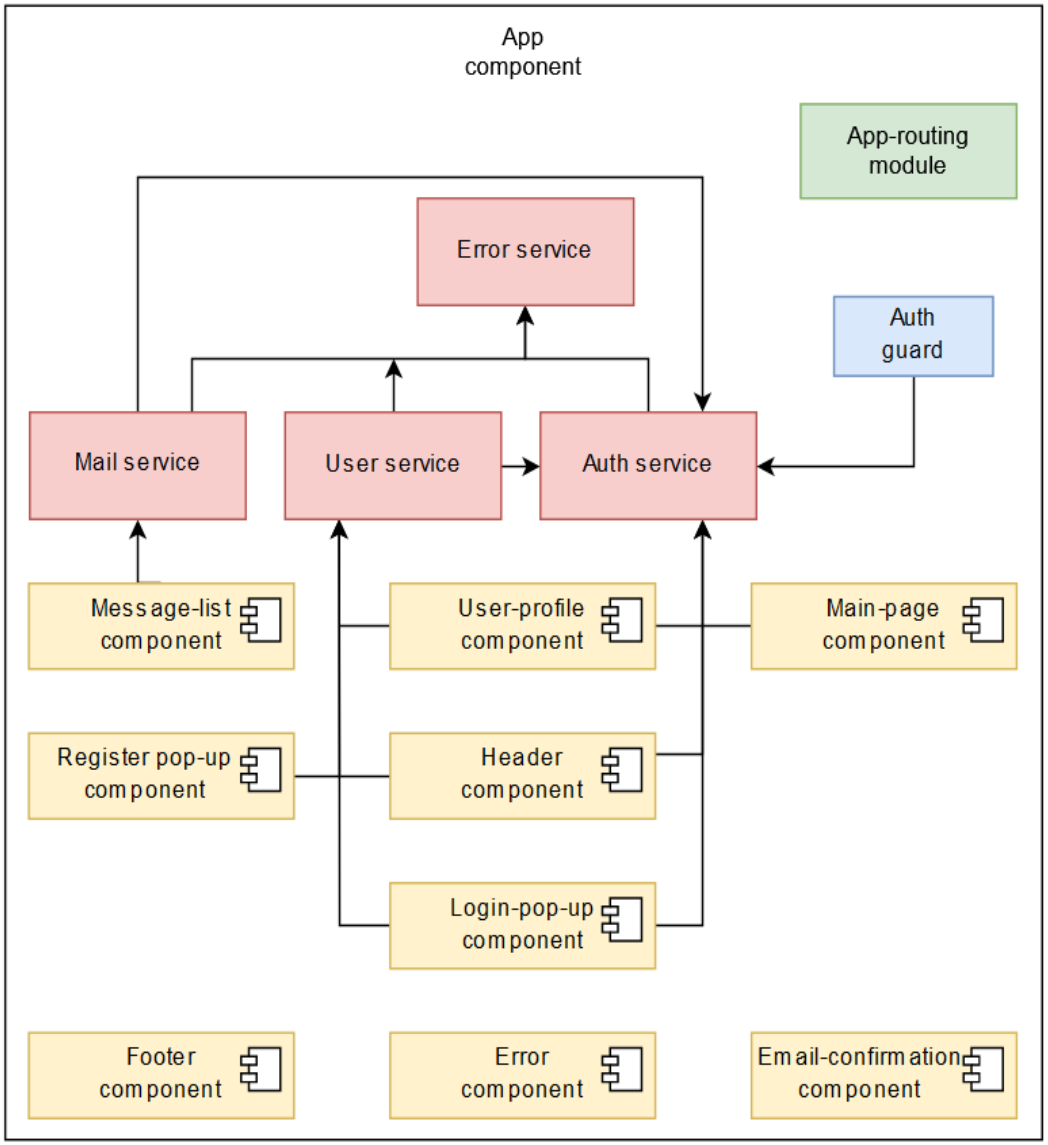
МОДУЛЬ MAIL_APP



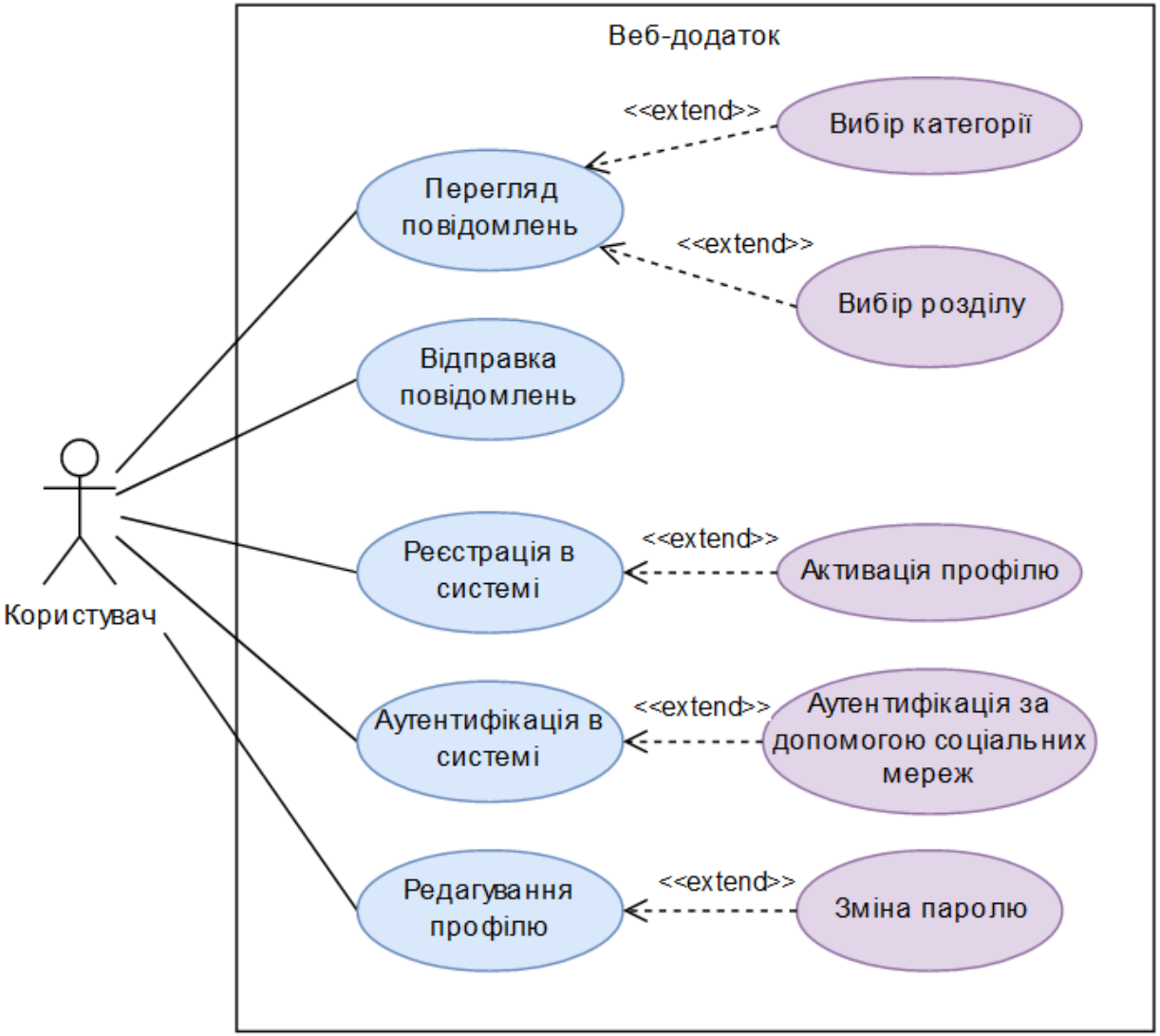
СТРУКТУРА БАЗИ ДАНИХ



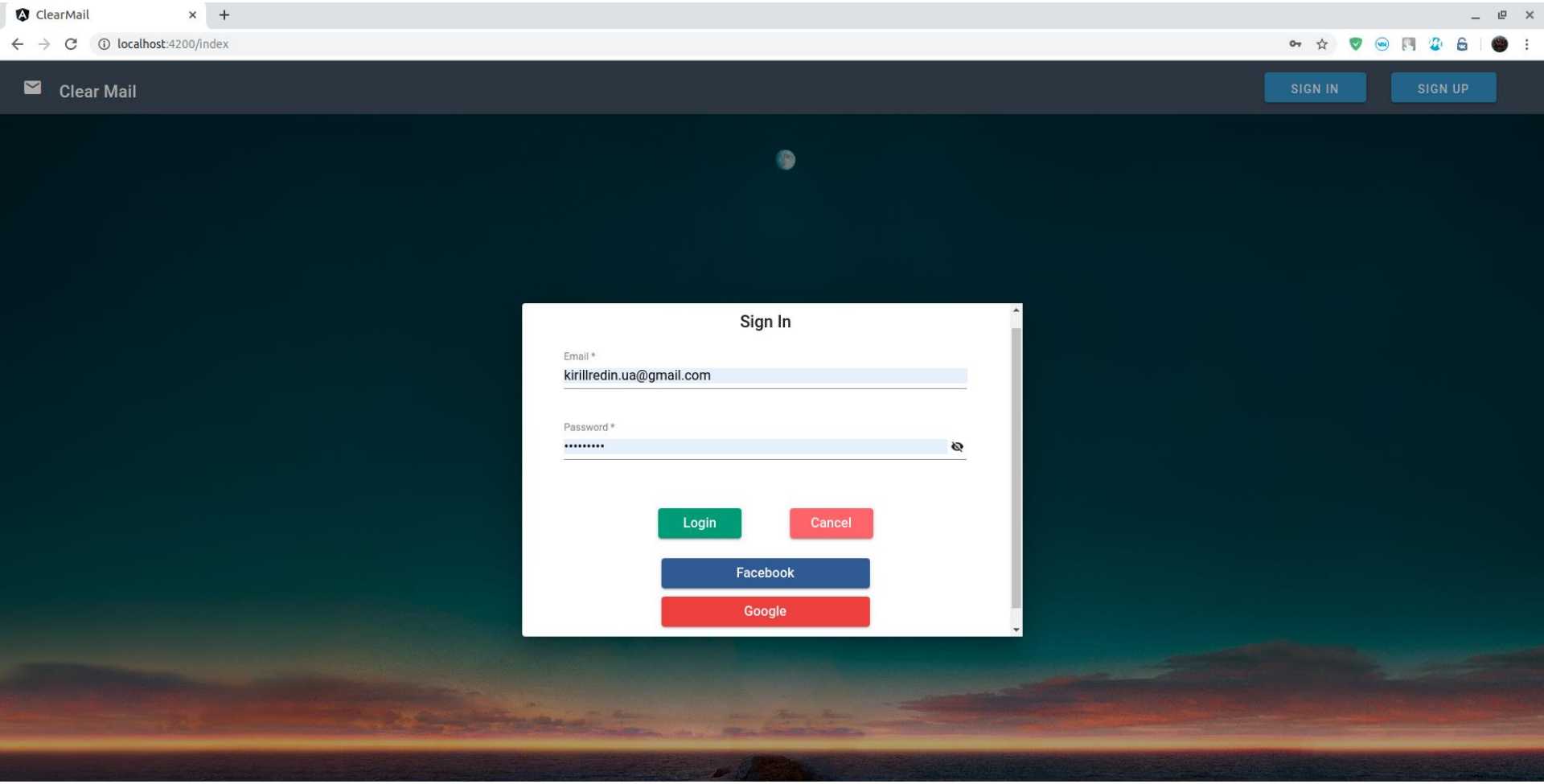
СТРУКТУРА КЛІЄНТСЬКОЇ ЧАСТИНИ



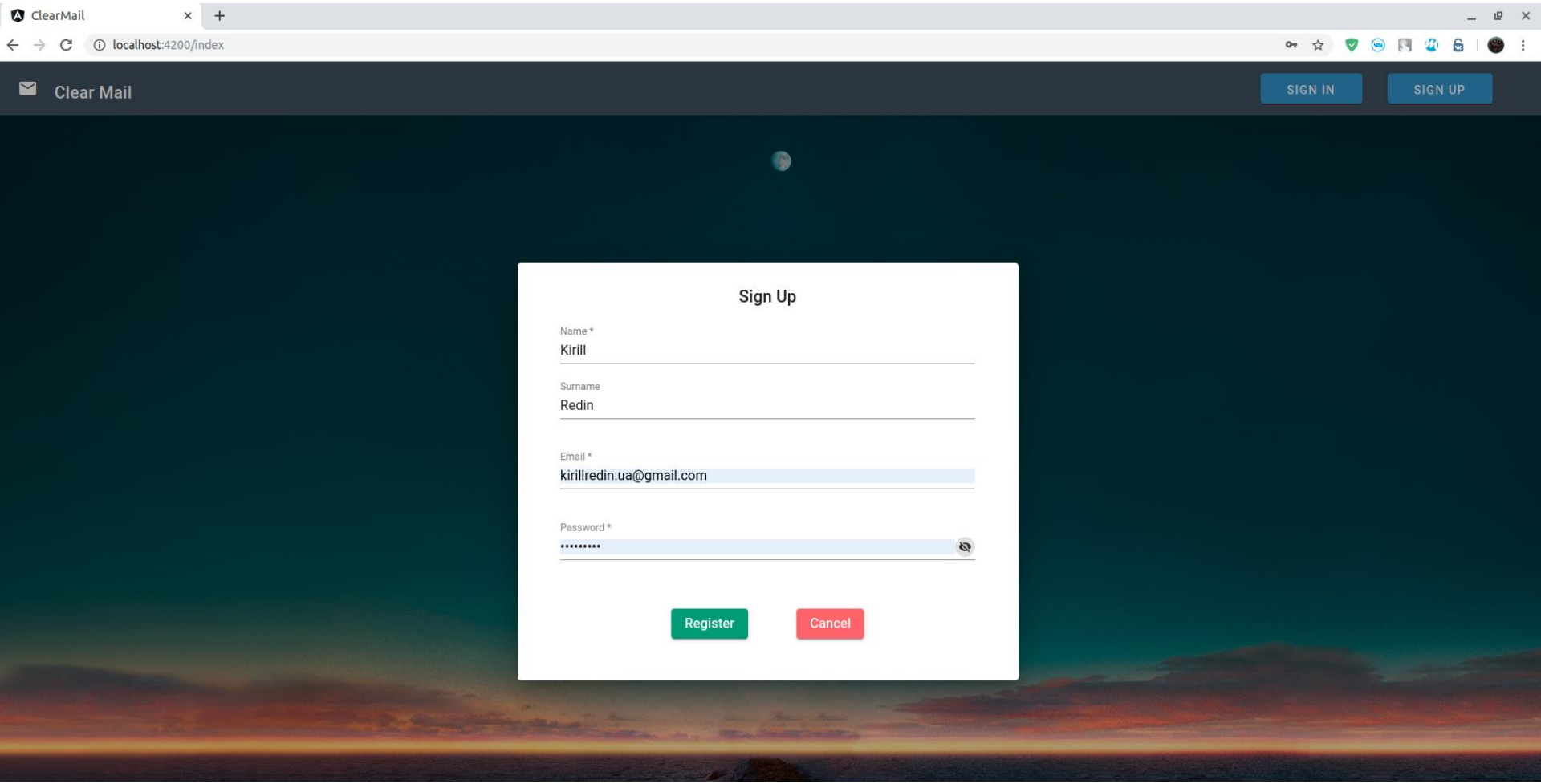
USE CASE ДІАГРАМА



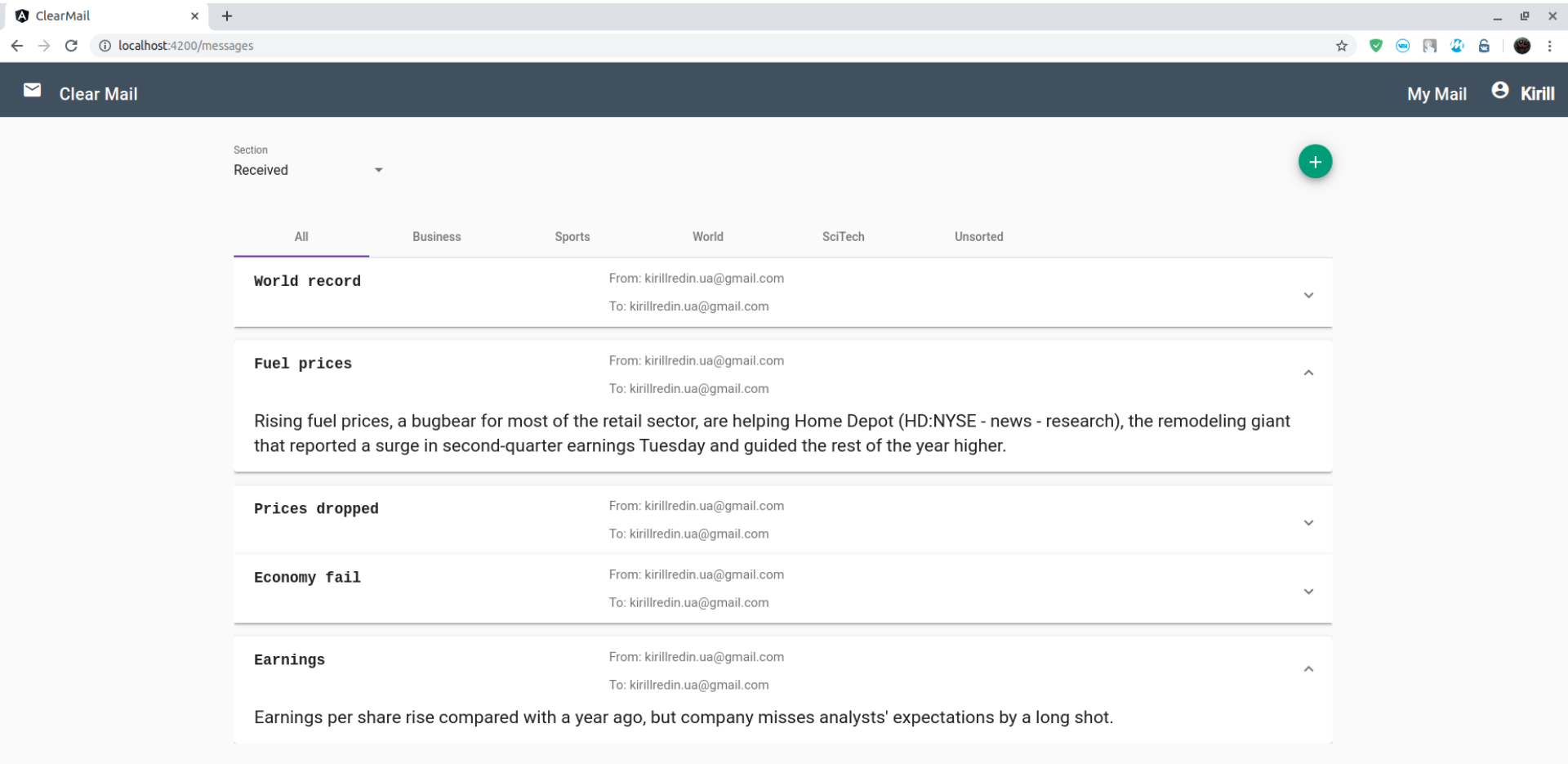
ГОЛОВНА СТОРІНКА ВЕБ-ДОДАТКУ



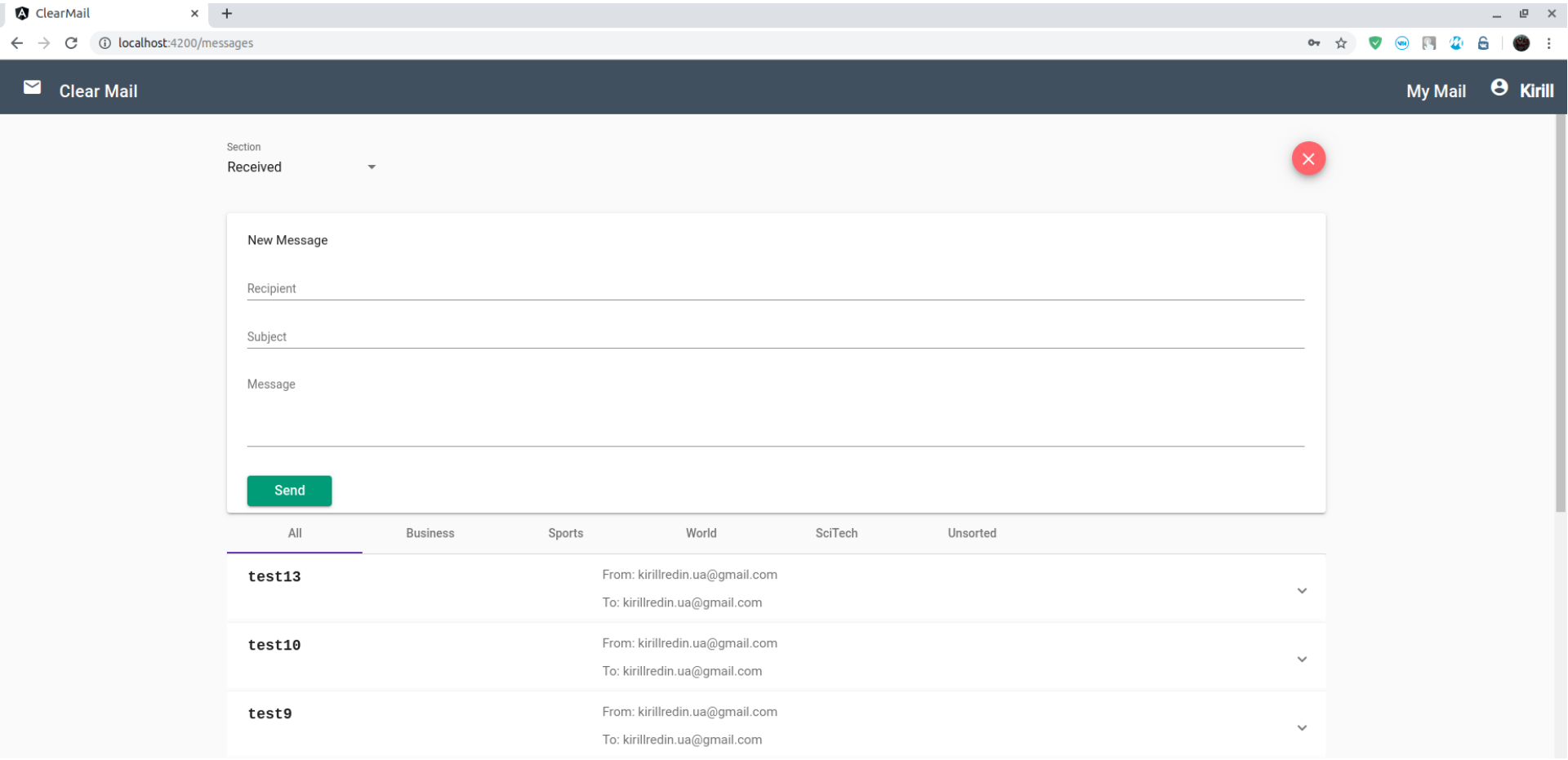
РЕЄСТРАЦІЯ КОРИСТУВАЧА



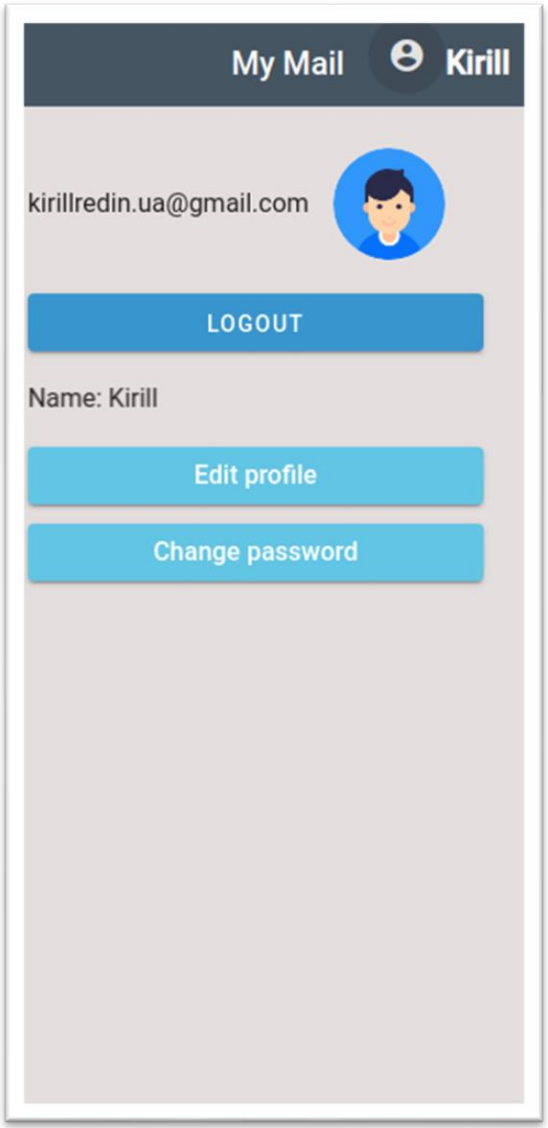
ПОВІДОМЛЕННЯ КОРИСТУВАЧА

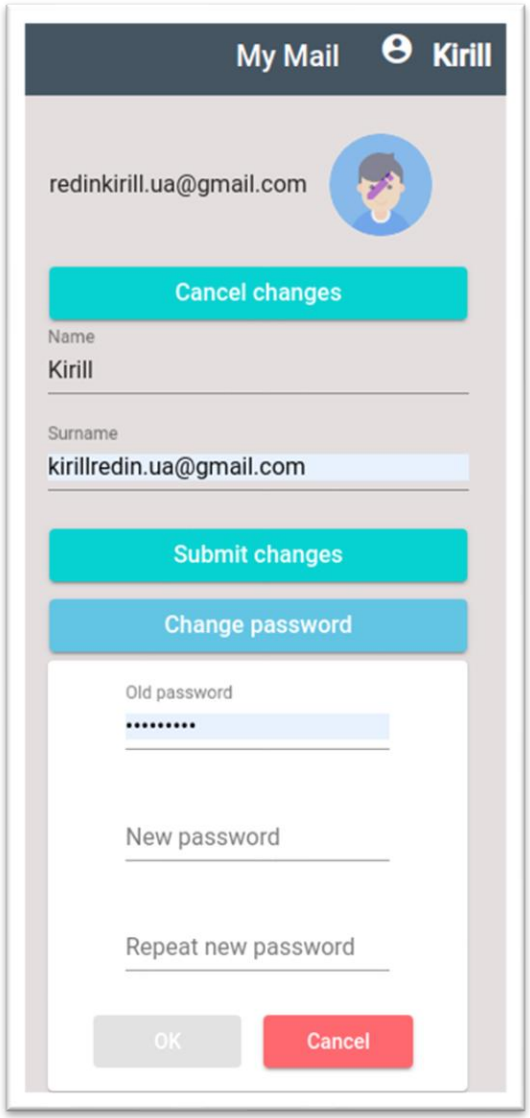


ВІДПРАВКА НОВОГО ПОВІДОМЛЕННЯ



ПРОФІЛЬ КОРИСТУВАЧА





ТЕСТУВАННЯ ФІЛЬТРАЦІЇ СПАМУ

	Розроблений класифікатор спаму	Дерево рішень	Випадковий ліс	Найближчий центроїд
Точність класифікації	0.9206	0.8742	0.8802	0.8060
Час навчання	44.8413 с.	2.7435 с.	4.559 с.	0.6587 с.
Час класифікації без серіалізації	44.8742 с.	2.7499 с.	4.5726 с.	0.6655 с.
Час класифікації З серіалізацією	0.6774 с.	0.5283 с.	0.5244 с.	0.5042

ТЕСТУВАННЯ КЛАСИФІКАЦІЇ ЗА КАТЕГОРІЯМИ

Граничне значення ймовірності	Розроблений класифікатор за категоріями	Дерево рішень	Випадковий ліс	Найближчий центроїд
0.7	0.8797	0.7393	0.8505	0.8011
0.75	0.8976	0.7393	0.8505	0.8011
0.8	0.8913	0.7393	0.8505	0.8011
Час навчання	1.4064 с.	11.7741 с.	18.4289 с.	0.6411 с.
Час класифікації без серіалізації	1.4136 с.	11.7799 с.	18.4411 с.	0.6476 с.
Час класифікації з серіалізацією	0.4483 с.	0.4555 с.	0.4785 с.	0.4793

БІЗНЕС-МОДЕЛЬ

<table>
<tr><td rowspan="2">Проблема
недостатня точність класифікації;
відсутність класифікації за декількома категоріями;
недосконаліть існуючих рішень.</td><td>Рішення
веб-додаток електронної пошти з можливістю групувати повідомлення за категоріями.</td><td rowspan="2">Унікальна ціннісна пропозиція
розроблений веб-додаток, який вирішує проблему неточної класифікації повідомлень завдяки реалізації комбінованого методу автоматизованої класифікації текстового вмісту повідомлень електронної пошти.</td><td rowspan="2">Ключові активності
створення документації;
розробка програмного продукту;
маркетинг;
супровід програмного продукту</td></tr>
<tr><td>Ключові метрики:
кількість проданих підписок
відгуки користувачів</td></tr>
<tr><td rowspan="2">Структура витрат
заробітні плати учасникам команди, юристу, бухгалтеру та прибиральнику;
аренда приміщення, обладнання;
маркетинг;
податки.</td><td>Ключові партнери
компанії-власники електронних поштових систем</td><td>Канали збуту
Власний магазин в веб додатку</td><td>Споживачі
англомовний сегмент користувачів електронних поштових систем</td></tr>
<tr><td colspan="3">Потоки доходів
продаж місячної та річної підписки на розширення списку категорій.</td></tr>
</table>

ФІНАНСОВИЙ ПЛАН СТАРТАПУ

	№ Місяця						Підсумок		№ Місяця						Підсумок
Прибутки / Витрати (тис. дол.)	1	2	3	4	5	6	Перше півріччя	Прибутки / Витрати (тис. дол.)	1	2	3	4	5	6	Друге півріччя
Продаж підписок	-	-	-	-	-	-	-	Продаж підписок	20	30	35	38	40	35	198
Зарплата	-9	-9	-9	-9	-9	-9	-54	Зарплата	-6	-6	-6	-6	-6	-6	-90
Оренда	-1.2	-1.2	-1.2	-1.2	-1.2.	-1.2	-7.2	Оренда	-0.8	-0.8	-0.8	-0.8	-0.8	-0.8	-12
Обладнання	-6	-	-	-	-	-	-6	Обладнання	-	-	-	-	-	-	-6
Маркетинг	-	-	-	-	-	-	-	Маркетинг	-1.2	-1.2	-1.2	-1.2	-	-	-4.8
Юрист	-0.7.	-0.7	-0.7	-0.7	-0.7	-0.7	-4.2	Юрист	-	-	-	-	-	-	-4.2
Бухгалтер	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-3.6	Бухгалтер	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-7.2
Прибирання	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-1.2	Прибирання	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-2.4
Прибутки – витрати	-11.7	-11.7	-11.7	-11.7	-11.7	-11.7	-76.2	Прибутки – витрати	11.2	21.2	26.2	29.2	32.4	27.4	71.4

НАУКОВО-ІНОВАЦІЙНА НОВИЗНА

- Запропоновано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти, що відрізняється від існуючих розділенням етапів фільтрації та класифікації за категоріями та дозволяє досягти підвищення точності класифікації.
- Запропоновано модифікацію метода Баєса, що відрізняється від існуючого методу введеним граничним значенням ймовірності, а також логарифмічної оцінки ймовірностей та дозволяє досягти підвищення точності класифікації та можливості класифікації текстового вмісту повідомлень за декількома категоріями.

ВИСНОВКИ

1. Проаналізовано існуючі методи класифікації текстового вмісту повідомлень, визначено їх основні переваги та недоліки для класифікації повідомлень електронної пошти.

2. Запропоновано комбінований метод автоматизованої класифікації текстового вмісту повідомлень електронної пошти з метою підвищення точності класифікації, а також надання можливості класифікації повідомлення за декількома категоріями.

3. Запропонований метод реалізовано у вигляді веб-додатку електронної пошти з можливістю групування повідомлень за категоріями. Обгрунтований вибір технологій, обраних для розробки веб-додатку. Описана архітектура розробленого веб-додатку, а також структура його серверної та клієнтської частин.

ВИСНОВКИ (продовження)

4. Проведено порівняння результатів класифікації комбінованим методом автоматизованої класифікації текстового вмісту повідомлень електронної пошти з іншими методами машинного навчання. Визначено, що реалізований метод надає більшу точність класифікації повідомлень ніж інші методи, використані для порівняння.

5. Виконаний аналіз економічної конкурентоспроможності реалізованого методу. За результатами аналізу сформовані таблиці прогнозованих витрат та прибутків від розробленого веб-додатку за перші дванадцять місяців, а також побудовано бізнес-модель.

АПРОБУВАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

1. XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2019).

Дякую за увагу!